图灵程序设计丛书

HTTP: The Definitive Guide

HTTP
权威指南



[美] David Gourley, Brian Totty, Marjorie Sayer, Sailu Reddy, Anshu Aggarwal 著

陈涓 赵振平 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

TURING 图灵程序设计丛书

HTTP权威指南

HTTP: The Definitive Guide

[美] David Gourley Brian Totty Marjorie Sayer
Sailu Reddy Anshu Aggarwal 著
陈涓 赵振平 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo
O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目（CIP）数据

HTTP权威指南 / (美) 古尔利 (Gourley,D.) 等著 ;
陈涓, 赵振平译. -- 北京 : 人民邮电出版社, 2012.8
（图灵程序设计丛书）（2012.10重印）
书名原文: HTTP:The Definitive Guide
ISBN 978-7-115-28148-7

Ⅰ. ①H… Ⅱ. ①古… ②陈… ③赵… Ⅲ. ①计算机
网络－通信协议 Ⅳ. ①TN915.04

中国版本图书馆CIP数据核字(2012)第095358号

内 容 提 要

本书是 HTTP 及其相关核心 Web 技术方面的权威著作，主要介绍了 Web 应用程序是如何工作的，核心的因特网协议如何与架构构建块交互，如何正确实现因特网客户端和服务器等。

本书适合所有想了解 HTTP 和 Web 底层结构的人阅读。

图灵程序设计丛书

HTTP权威指南

◆ 著　[美] David Gourley Brian Totty Marjorie Sayer Sailu Reddy Anshu Aggarwal
译　陈 涓 赵振平
责任编辑　李 盼
执行编辑　丁晓昀

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：45
字数：1067千字　2012年 10 月第 1 版
印数：5－8 000册　2012年 10 月北京第 2 次印刷

著作权合同登记号　图字：01-2011-1852号

ISBN 978-7-115-28148-7

定价：109.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

版权声明

O'Reilly Media, Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O'Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会聚集了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O'Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar 博客有口皆碑。”

——Wired

“O'Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

前言

HTTP（Hypertext Transfer Protocol，超文本传输协议[①]）是在万维网上进行通信时所使用的协议方案。HTTP 有很多应用，但最著名的是用于 Web 浏览器和 Web 服务器之间的双工通信。

HTTP 起初是一个简单的协议，因此你可能会认为关于这个协议没有太多好说的。但现在，你手上拿着的却是一本将近两斤重的书。如果你想知道我们怎么会写出一本 700 多页的关于 HTTP 的书，就去看看目录吧。本书不仅仅是一本 HTTP 首部参考手册，它还是一本名副其实的 Web 架构“圣经”。

本书中，我们会将 HTTP 中一些互相关联且常被误解的规则梳理清楚，并编写了一系列基于各种主题的章节介绍 HTTP 各方面的特性。纵观全书，我们对 HTTP“为什么”这样做进行了详细的解释，而不仅仅停留在它是“怎么做”的。而且，为了节省大家寻找参考文献的时间，我们还介绍了很多 HTTP 应用程序正常工作所必需且重要的非 HTTP 技术。在条理清晰的附录中，可以找到按照字母排序的首部参考（这些首部构成了最常见的 HTTP 文本的基础）。我们希望这种概念性的设计有助于读者更好地使用 HTTP。

本书是为所有希望理解 HTTP 和 Web 底层结构的人编写的。软硬件工程师也可以将本书作为 HTTP 及相关 Web 技术参考书使用。系统架构师和网络管理员可以通过本书更好地了解如何设计、实现并管理复杂的网络架构。性能工程师和分析人员可以从缓存和性能优化的相关章节中获益。市场营销和咨询专家还可以通过概念介绍更好地理解 Web 技术的前景。

① HTTP 译为“超文本传输协议”，其中“transfer”使用了“传输”的含义，但依据 HTTP 制定者之一 Roy Fielding 博士的论文，“transfer”表示的是“（状态的）转移”，而不是“传输”。怎样翻译才更符合 HTTP 的原意，其讨论可参见图灵社区的文章，地址是 ituring.com.cn/article/details/1817。

本书澄清了一些常见的误解，推荐了“各种业内诀窍”，提供了便捷的参考资料，并且用通俗易懂的语言阐述了枯燥且令人费解的标准规范，还详细探讨了 Web 正常工作所必需且互相关联的技术。

本书创作历时良久，是由很多热衷于因特网技术的人共同完成的，希望它能对你有所帮助。

运行实例：Joe的五金商店

本书的很多章节都涉及了一个假想的在线五金与家装商店示例，通过这个“Joe 的五金商店”来说明一些技术概念。我们为这个商店构建了一个真实的 Web 站点（http://www.joes-hardware.com），以便大家能够测试书中的部分实例。只要本书仍在销售，我们就会一直维护好这个 Web 站点。

本书内容

本书包含 21 章，分为 5 个逻辑部分（每部分都是一个技术专题），以及 8 个很有用的附录，这些附录包含了参考资料，以及对相关技术的介绍。

第一部分　HTTP：Web 的基础
第二部分　HTTP 结构
第三部分　识别、认证与安全
第四部分　实体、编码和国际化
第五部分　内容发布与分发
第六部分　附录

第一部分用 4 章的篇幅描述了 Web 的基础构件与 HTTP 的核心技术。

- 第 1 章简要介绍了 HTTP。
- 第 2 章详细阐述了统一资源定位符（Uniform Resource Locator，URL）的格式，以及 URL 在因特网上命名的各种类型的资源，还介绍了统一资源名（Uniform Resource Name，URN）的演变过程。
- 第 3 章详细介绍了 HTTP 报文是如何传送 Web 内容的。
- 第 4 章解释了 HTTP 连接管理过程中一些经常会引起误解且少有文档说明的规则和行为。

第二部分重点介绍了 Web 系统的结构构造块：HTTP 服务器、代理、缓存、网关以及机器人应用程序。（当然，Web 浏览器也是一种构造块，但在本书的第一部分已经对其进行过很详细的介绍了。）第二部分包含以下 6 章。

- 第 5 章简要介绍了 Web 服务器结构。
- 第 6 章深入研究了 HTTP 代理服务器，HTTP 代理服务器是作为 HTTP 服务与控制平台使用的中间服务器。
- 第 7 章深入研究了 Web 缓存的问题。缓存是通过保存常用文档的本地副本来提高性能、减少流量的设备。
- 第 8 章探讨了网关和应用服务器的概念，通过它们，HTTP 就可以与使用不同协议（包括 SSL 加密协议）的软件进行通信了。
- 第 9 章介绍了 Web 上的各种客户端类型，包括无处不在的浏览器、机器人和网络蜘蛛以及搜索引擎。
- 第 10 章讲述了仍在研究之中的 HTTP 协议：HTTP-NG 协议。

第三部分提供了一套用于追踪身份、增强安全性以及控制内容访问的技术和技巧。包含下列 4 章。

- 第 11 章讨论了一些识别用户的技术，以便向用户提供私人化的内容服务。
- 第 12 章重点介绍了一些验证用户身份的基本方式。这一章还对 HTTP 认证机制与数据库的接口问题进行了研究。
- 第 13 章详述了摘要认证，它是对 HTTP 的建议性综合增强措施，可以大幅度提高其安全性。
- 第 14 章说明了因特网的密码体系、数字证书以及 SSL。

第四部分涵盖 HTTP 报文主体和 Web 标准，前者包含实际内容，后者描述并处理主体内容。第四部分包含以下 3 章。

- 第 15 章介绍了 HTTP 内容的结构。
- 第 16 章探讨了一些 Web 标准，通过这些标准，全球范围内的用户都可以交换以不同语言和字符集表示的内容。
- 第 17 章解释了一些用于协商可接受内容的机制。

第五部分介绍了发布和传播 Web 内容的技巧。包括以下 4 章。

- 第 18 章讨论了在现代的网站托管环境中布署服务器的方式以及 HTTP 对虚拟网站托管的支持。
- 第 19 章探讨了一些创建 Web 内容，并将其装载到 Web 服务器中去的技术。
- 第 20 章介绍了能够将输入 Web 流量分散到一组服务器上去的一些工具和技术。
- 第 21 章介绍了一些日志格式和常见问题。

第六部分是一些很有用的参考附录，以及相关技术的教程。

- 附录 A 详述了统一资源描述符（Uniform Resource Identifier，URI）方案所支持的协议。

- 附录 B 列出了 HTTP 的响应代码，方便使用。
- 附录 C 提供了 HTTP 首部字段的参考列表。
- 附录 D 列出了大量的 MIME 类型，解释了 MIME 类型的注册方式。
- 附录 E 介绍了 HTTP 认证中使用的 Base-64 编码。
- 附录 F 详述了如何实现 HTTP 中的各种认证方案。
- 附录 G 定义了 HTTP 首部的语言标签值。
- 附录 H 列出了用以支持国际化 HTTP 的字符编码。

每章都包含很多实例，以及到其他相关的参考资料的链接。

排版约定

本书使用了下列排版约定。

- 楷体

用于 URL、C 函数、命令名、MIME 类型、新术语的定义以及重点内容。

- 等宽字体

用于计算机的输出、代码以及所有文字文本。

- 加粗等宽字体

用于用户的输入。

意见及问题

请将有关此书的意见及问题发给出版商：

美国：
O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
中国：
北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

本书有一个 Web 页面，上面列出了勘误表、一些实例以及所有的附加信息。可以通过以下链接来访问这个页面。

http://www.oreilly.com/catalog/httptdg/

为本书提意见或者询问一些技术性问题，可以向以下地址发送邮件。

bookquestions@oreilly.com

更多与书籍、会议、资源中心以及 O'Reilly 网络有关的问题，都请参见 O'Reilly 的网站。

http://www.oreilly.com

致谢

本书是很多人共同劳动的成果。五位作者要感谢一些人，感谢他们为这本书所作出的巨大贡献。

首先，我们要感谢 O'Reilly 的编辑 Linda Mui。Linda 早在 1996 年就与 David 和 Brian 进行了首次接触，她还提炼了几个概念，并将其融入到今天大家拿到的这本书中。Linda 还帮助我们这帮首次写书、徘徊不定的人协调一致地按计划逐步完成了这本书的写作（尽管我们完成得并不怎么快）。最重要的是，Linda 给了我们一个创作此书的机会。我们要对她表示由衷的感谢。

我们还要感谢以下人士，他们非常聪明博学而且非常友善，为校对、注释并修订本书草稿花费了大量精力。他们是：Tony Bourke、Sean Burke、Mike Chowla、Shernaz Daver、Fred Douglis、Paula Ferguson、Vikas Jha、Yves Lafon、Peter Mattis、Chuck Neerdaels、Luis Tavera、Duane Wessels、Dave Wu 和 Marco Zagha。他们的一些观点和建议大大提升了本书的质量。

本书大部分精美的插图都是由 O'Reilly 的 Rob Romano 制作的。为了能够更加清晰地描述一些微妙的概念，本书使用了大量详尽备至的插图。其中很多插图制作起来都很费劲，而且还经过了大量的修改。如果一幅图相当于一千个字的话，Rob 就相当于为本书增加了数百页的篇幅。

Brian 还要特别感谢所有作者对本项目的付出。为了对 HTTP 作出首次详细而又切实可行的剖析，作者们投入了大量的时间。其间虽然出现了婚礼、孩子出世、刻不容缓的工作项目、创业公司起步以及就读研究生院等诸多问题，但作者们的共同努力使这个项目得以圆满完成。我们相信，每个人的努力付出都是值得的，而且最重要的是，这项工作为大家提供了一项很有价值的服务。Brian 还要感谢 Inktomi 的员工们，感谢他们的热情和支持，以及他们对 HTTP 在实际应用程序中应用状况的深

刻洞察力。同时，还要感谢 Cajun-shop.com 允许我们使用他们的站点来展示书中的一些范例。

David 要感谢他的家人，尤其是母亲和祖父长期以来不懈的支持。他要感谢那些在写书这几年中忍受他古怪作息习惯的家人们。他要对 Slurp、Orctomi 和 Norma 所做的一切表示感谢，还要感谢合作者们的辛勤工作。最后，他要感谢 Brian 说服自己再次冒险。

Marjorie 要感谢她丈夫 Alan Liu 的技术洞察力，以及他对家庭的支持和理解。Marjorie 还要感谢合作者们丰富且深刻的灵感和洞察力。在编写过程中能够与他们共同工作，她感到非常开心。

Sailu 要感谢 David 和 Brian 为他提供机会参与编写本书，感谢 Chuck Neerdaels 将他引入了 HTTP 的世界。

Anshu 要感谢他的妻子 Rashi 和他的父母。尽管本书的编写旷日持久，但家人依旧对他有着足够的耐心，不断地支持并鼓励他。

最后，作者们要集体感谢各位著名和无名的因特网先驱们，他们在过去 40 年中所做的研究、开发和传播工作对我们的科学界、社会及经济团体作出了巨大的贡献。没有他们的工作，就没有本书所要讨论的话题。

目录

第一部分　HTTP：Web 的基础

第二部分 HTTP 结构

第三部分 识别、认证与安全

第四部分 实体、编码和国际化

第五部分　内容发布与分发

第六部分 附 录

第一部分

HTTP：Web的基础

本部分主要概述 HTTP 协议。接下来的 4 章介绍了 Web 的基础构件以及 HTTP 的核心技术。

- 第 1 章简要概述 HTTP。
- 第 2 章详细介绍了 URL 的格式，以及 URL 在因特网上命名的各种类型的资源，并对其向 URN 的发展作了概要介绍。
- 第 3 章详细说明了用来传输 Web 内容的 HTTP 报文。
- 第 4 章讨论了一些通过 HTTP 管理 TCP 连接时常被误解且很少有文档说明的规则和行为。

第1章

HTTP概述

Web 浏览器、服务器和相关的 Web 应用程序都是通过 HTTP 相互通信的。HTTP 是现代全球因特网中使用的公共语言。

本章是对 HTTP 的简要介绍。在本章中可以看到 Web 应用程序是如何使用 HTTP 进行通信的，这样就可以对 HTTP 如何完成其工作有个大概印象。我们将特别介绍以下方面的内容：

- Web 客户端与服务器是如何通信的；
- （表示 Web 内容的）资源来自何方；
- Web 事务是怎样工作的；
- HTTP 通信所使用的报文格式；
- 底层 TCP 网络传输；
- 不同的 HTTP 协议变体；
- 因特网上安装的大量 HTTP 架构组件中的一部分。

我们有很多话题要讨论，就此开始 HTTP 之旅吧。

1.1 HTTP——因特网的多媒体信使

每天，都有数以亿万计的 JPEG 图片、HTML 页面、文本文件、MPEG 电影、WAV 音频文件、Java 小程序和其他资源在因特网上游弋。HTTP 可以从遍布全世界的 Web 服务器上将这些信息块迅速、便捷、可靠地搬移到人们桌面上的 Web 浏览器上去。

HTTP 使用的是可靠的数据传输协议，因此即使数据来自地球的另一端，它也能够确保数据在传输的过程中不会被损坏或产生混乱。这样，用户在访问信息时就不用担心其完整性了，因此对用户来说，这是件好事。而对因特网应用程序开发人员来
1～3 说也同样如此，因为这样就无需担心 HTTP 通信会在传输过程中被破坏、复制或产生畸变了。开发人员可以专注于应用程序特有细节的编写，而不用考虑因特网中存在的一些缺陷和问题。

下面，就让我们来近距离地观察一下 HTTP 是如何传输 Web 流量的。

1.2 Web客户端和服务器

Web 内容都是存储在 Web 服务器上的。Web 服务器所使用的是 HTTP 协议，因此经常会被称为 HTTP 服务器。这些 HTTP 服务器存储了因特网中的数据，如果

HTTP 客户端发出请求的话，它们会提供数据。客户端向服务器发送 HTTP 请求，服务器会在 HTTP 响应中回送所请求的数据，如图 1-1 所示。HTTP 客户端和 HTTP 服务器共同构成了万维网的基本组件。

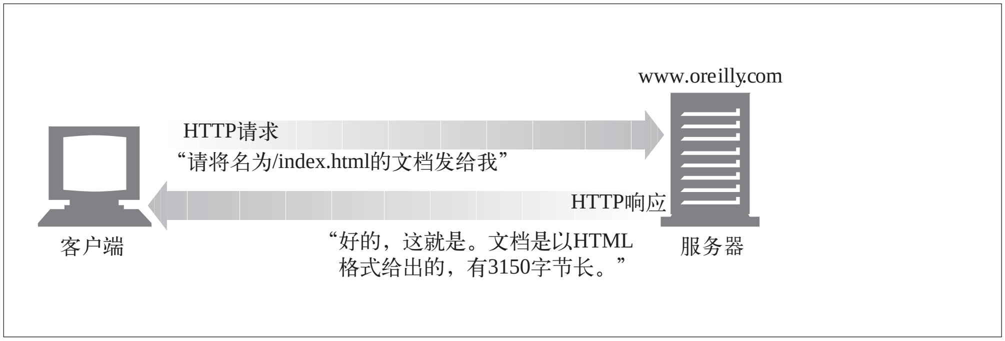


图 1-1　Web 客户端和服务器

可能你每天都在使用 HTTP 客户端。最常见的客户端就是 Web 浏览器，比如微软的 Internet Explorer 或网景的 Navigator。Web 浏览器向服务器请求 HTTP 对象，并将这些对象显示在你的屏幕上。

浏览一个页面时（比如 http://www.oreilly.com/index.html），浏览器会向服务器 www.oreilly.com 发送一条 HTTP 请求（参见图 1-1）。服务器会去寻找所期望的对象（在这个例子中就是 /index.html），如果成功，就将对象、对象类型、对象长度以及其他一些信息放在 HTTP 响应中发送给客户端。

1.3　资源

Web 服务器是 Web *资源*（Web resource）的宿主。Web 资源是 Web 内容的源头。最简单的 Web 资源就是 Web 服务器文件系统中的静态文件。这些文件可以包含任意内容：文本文件、HTML 文件、微软的 Word 文件、Adobe 的 Acrobat 文件、JPEG 图片文件、AVI 电影文件，或所有其他你能够想到的格式。

但资源不一定非得是静态文件。资源还可以是根据需要生成内容的软件程序。这些动态内容资源可以根据你的身份、所请求的信息或每天的不同时段来产生内容。它
们可以为你显示照相机中活生生的照片，也可以帮你进行股票交易，搜索房产数据 4
库，或者从在线商店中购买礼物（参见图 1-2）。

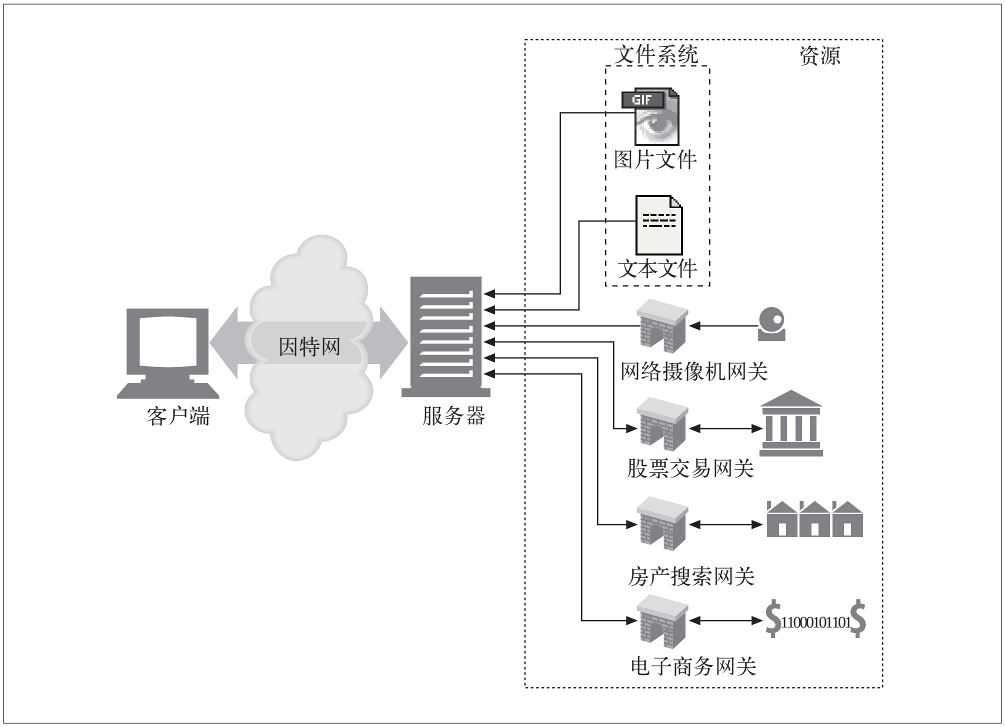


图 1-2　所有能够提供 Web 内容的东西都是 Web 资源

总之，所有类型的内容来源都是资源。包含公司销售预测电子表格的文件是一种资源。扫描本地公共图书馆书架的 Web 网关是一种资源。因特网搜索引擎也是一种资源。

1.3.1　媒体类型

因特网上有数千种不同的数据类型，HTTP 仔细地给每种要通过 Web 传输的对象都打上了名为 MIME 类型（MIME type）的数据格式标签。最初设计 MIME（Multipurpose Internet Mail Extension，多用途因特网邮件扩展）是为了解决在不同的电子邮件系统之间搬移报文时存在的问题。MIME 在电子邮件系统中工作得非常好，因此 HTTP 也采纳了它，用它来描述并标记多媒体内容。

Web 服务器会为所有 HTTP 对象数据附加一个 MIME 类型（参见图 1-3）。当 Web 浏览器从服务器中取回一个对象时，会去查看相关的 MIME 类型，看看它是否知道应该如何处理这个对象。大多数浏览器都可以处理数百种常见的对象类型：显示图片文件、解析并格式化 HTML 文件、通过计算机声卡播放音频文件，或者运行外部
5 插件软件来处理特殊格式的数据。

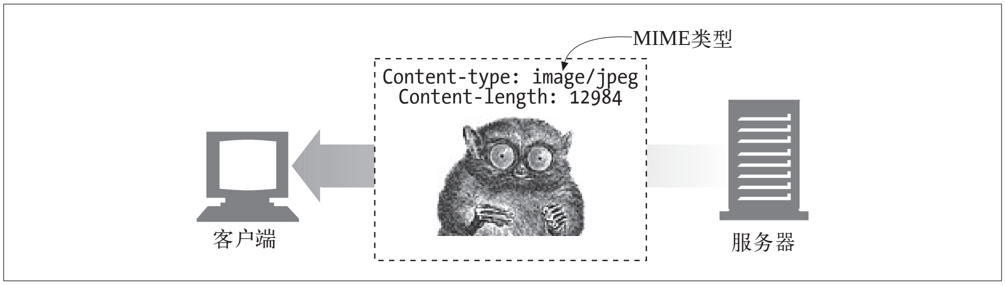


图 1-3　与数据内容一同回送的 MIME 类型

MIME 类型是一种文本标记，表示一种主要的对象类型和一个特定的子类型，中间由一条斜杠来分隔。

- HTML 格式的文本文档由 `text/html` 类型来标记。
- 普通的 ASCII 文本文档由 `text/plain` 类型来标记。
- JPEG 格式的图片为 `image/jpeg` 类型。
- GIF 格式的图片为 `image/gif` 类型。
- Apple 的 QuickTime 电影为 `video/quicktime` 类型。
- 微软的 PowerPoint 演示文件为 `application/vnd.ms-powerpoint` 类型。

常见的 MIME 类型有数百个，实验性或用途有限的 MIME 类型则更多。附录 D 提供了一个非常完整的 MIME 类型列表。

1.3.2　URI

每个 Web 服务器资源都有一个名字，这样客户端就可以说明它们感兴趣的资源是什么了。服务器资源名被称为*统一资源标识符*（Uniform Resource Identifier，URI）。URI 就像因特网上的邮政地址一样，在世界范围内唯一标识并定位信息资源。

这是 Joe 的五金商店的 Web 服务器上一个图片资源的 URI：

http://www.joes-hardware.com/specials/saw-blade.gif

图 1-4 显示了 URI 是怎样指示 HTTP 协议去访问 Joe 商店服务器上的图片资源的。给定了 URI，HTTP 就可以解析出对象。URI 有两种形式，分别称为 URL 和 URN。现在我们分别来看看这些资源标识符类型。

1.3.3　URL

统一资源定位符（URL）是资源标识符最常见的形式。URL 描述了一台特定服务器上某资源的特定位置。它们可以明确说明如何从一个精确、固定的位置获取资源。

6 图 1-4 显示了 URL 如何精确地说明某资源的位置以及如何去访问它。表 1-1 显示了几个 URL 实例。

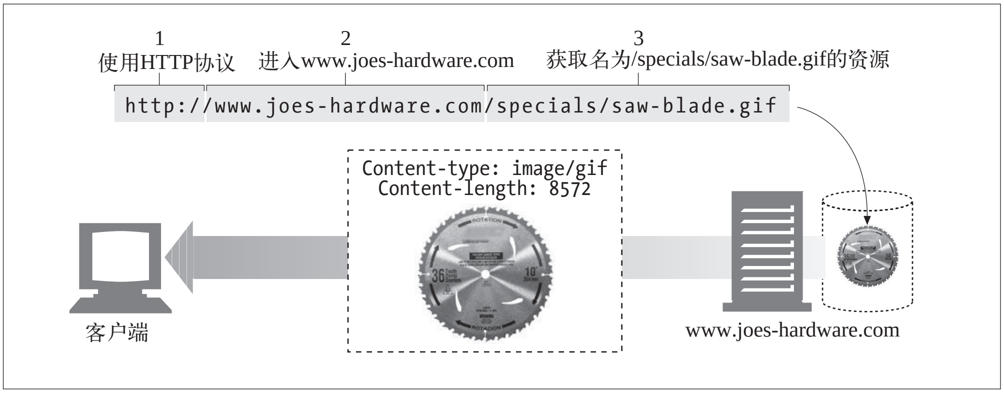


图 1-4 URL 说明了协议、服务器和本地资源

表1-1 URL实例

URL	描　　述
http://www.oreilly.com/index.html	O'Reilly & Associates 公司的主 URL
http://www.yahoo.com/images/logo.gif	Yahoo! 的 Web 站点标志 URL
http://www.joes-hardware.com/inventory-check.cgi?item=12731	一个查看库存条目 #12731 是否有现货的程序的 URL
ftp://joe:tools4u@ftp.joes-hardware.com/locking-pliers.gif	以密码保护的 FTP 作为访问协议的 locking-pliers.gif 图片文件的 URL

大部分 URL 都遵循一种标准格式，这种格式包含三个部分。

- URL 的第一部分被称为方案（scheme），说明了访问资源所使用的协议类型。这部分通常就是 HTTP 协议（http://）。
- 第二部分给出了服务器的因特网地址（比如，www.joes-hardware.com）。
- 其余部分指定了 Web 服务器上的某个资源（比如，/specials/saw-blade.gif）。

现在，几乎所有的 URI 都是 URL。

1.3.4 URN

URI 的第二种形式就是统一资源名（URN）。URN 是作为特定内容的唯一名称使用的，与目前的资源所在地无关。使用这些与位置无关的 URN，就可以将资源四处搬移。通过 URN，还可以用同一个名字通过多种网络访问协议来访问资源。

比如，不论因特网标准文档 RFC 2141 位于何处（甚至可以将其复制到多个地方），都可以用下列 URN 来命名它： 7

urn:ietf:rfc:2141

URN 仍然处于试验阶段，还未大范围使用。为了更有效地工作，URN 需要一个支撑架构来解析资源的位置。而此类架构的缺乏也延缓了其被采用的进度。但 URN 确实为未来发展作出了一些令人兴奋的承诺。我们将在第 2 章较为详细地讨论 URN，而本书的其余部分讨论的基本上都是 URL。

除非特殊说明，否则本书的其余部分都会使用约定的术语，并且会不加区别地使用 URI 和 URL。

1.4 事务

我们来更仔细地看看客户端是怎样通过 HTTP 与 Web 服务器及其资源进行事务处理的。一个 HTTP 事务由一条（从客户端发往服务器的）请求命令和一个（从服务器发回客户端的）响应结果组成。这种通信是通过名为 HTTP 报文（HTTP message）的格式化数据块进行的，如图 1-5 所示。

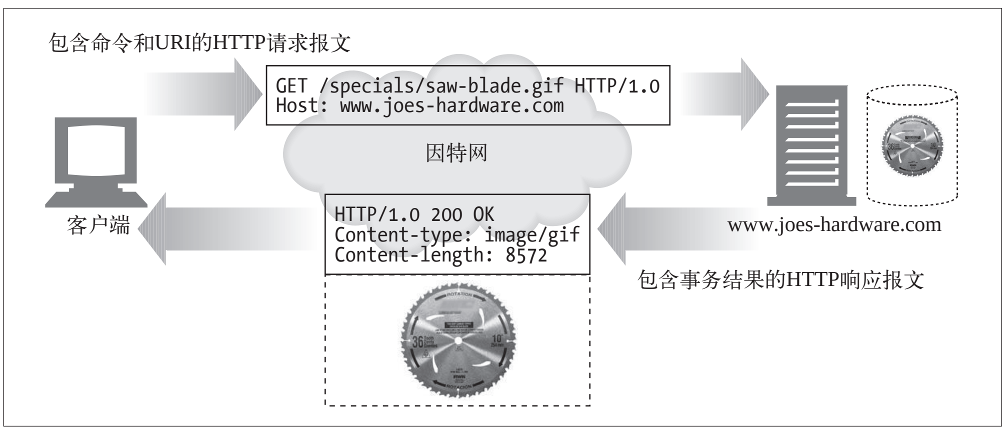


图 1-5 包含请求及响应报文的 HTTP 事务

1.4.1 方法

HTTP 支持几种不同的请求命令，这些命令被称为 HTTP *方法*（HTTP method）。每条 HTTP 请求报文都包含一个方法。这个方法会告诉服务器要执行什么动作（获取一个 Web 页面、运行一个网关程序、删除一个文件等）。表 1-2 列出了五种常见的 HTTP 方法。

表1-2　一些常见的HTTP方法

HTTP方法	描　　述
GET	从服务器向客户端发送命名资源
PUT	将来自客户端的数据存储到一个命名的服务器资源中去
DELETE	从服务器中删除命名资源
POST	将客户端数据发送到一个服务器网关应用程序
HEAD	仅发送命名资源响应中的 HTTP 首部

8

我们会在第 3 章详细讨论 HTTP 方法。

1.4.2　状态码

每条 HTTP 响应报文返回时都会携带一个状态码。状态码是一个三位数字的代码，告知客户端请求是否成功，或者是否需要采取其他动作。表 1-3 显示了几种常见的状态码。

表1-3　一些常见的HTTP状态码

HTTP状态码	描　　述
200	OK。文档正确返回
302	Redirect（重定向）。到其他地方去获取资源
404	Not Found（没找到）。无法找到这个资源

伴随着每个数字状态码，HTTP 还会发送一条解释性的“原因短语”文本（参见图 1-5 中的响应报文）。包含文本短语主要是为了进行描述，所有的处理过程使用的都是数字码。

HTTP 软件处理下列状态码和原因短语的方式是一样的。

```
200 OK
200 Document attached
200 Success
200 All’s cool, dude
```

第 3 章详细解释了 HTTP 状态码。

1.4.3　Web页面中可以包含多个对象

应用程序完成一项任务时通常会发布多个 HTTP 事务。比如，Web 浏览器会发布一系列 HTTP 事务来获取并显示一个包含了丰富图片的 Web 页面。浏览器会执行一个事务来获取描述页面布局的 HTML“框架”，然后发布另外的 HTTP 事务来获取每个嵌入式图片、图像面板、Java 小程序等。这些嵌入式资源甚至可能位于不同的服

务器上，如图 1-6 所示。因此，一个“Web 页面”通常并不是单个资源，而是一组资源的集合。 9

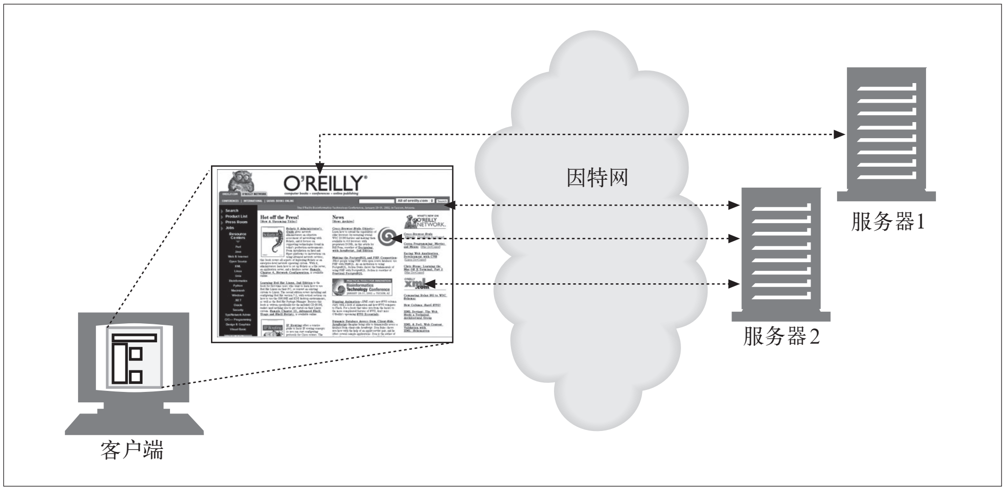


图 1-6　复合 Web 页面要为每个嵌入式资源使用一个单独的 HTTP 事务

1.5　报文

现在我们来快速浏览一下 HTTP 请求和响应报文的结构。第 3 章会深入研究 HTTP 报文。

HTTP 报文是由一行一行的简单字符串组成的。HTTP 报文都是纯文本，不是二进制代码，所以人们可以很方便地对其进行读写[1]。图 1-7 显示了一个简单事务所使用的 HTTP 报文。

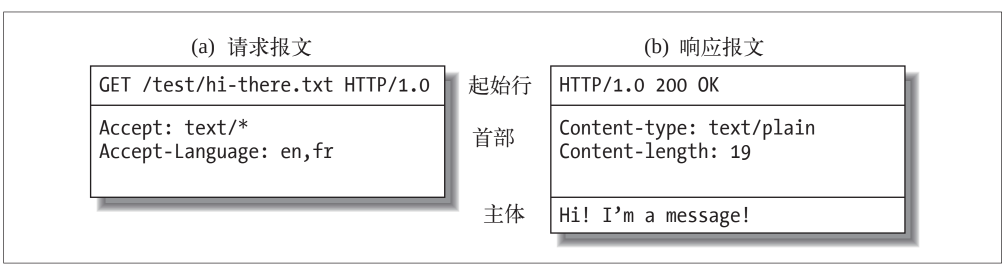


图 1-7　由一行行的简单文本结构组成的 HTTP 报文

注 1：有些程序员会抱怨 HTTP 的语法解析太困难了，这项工作需要很多技巧，而且很容易出错，尤其是在设计高速软件的时候更是如此。二进制格式或更严格的文本格式可能更容易处理，但大多数 HTTP 程序员都很欣赏 HTTP 的可扩展性以及可调试性。

从 Web 客户端发往 Web 服务器的 HTTP 报文称为*请求报文*（request message）。从服务器发往客户端的报文称为*响应报文*（response message），此外没有其他类型的
10 HTTP 报文。HTTP 请求和响应报文的格式很类似。

HTTP 报文包括以下三个部分。

- *起始行*
  报文的第一行就是起始行，在请求报文中用来说明要做些什么，在响应报文中说明出现了什么情况。

- *首部字段*
  起始行后面有零个或多个首部字段。每个首部字段都包含一个名字和一个值，为了便于解析，两者之间用冒号（:）来分隔。首部以一个空行结束。添加一个首部字段和添加新行一样简单。

- *主体*
  空行之后就是可选的报文主体了，其中包含了所有类型的数据。请求主体中包括了要发送给 Web 服务器的数据；响应主体中装载了要返回给客户端的数据。起始行和首部都是文本形式且都是结构化的，而主体则不同，主体中可以包含任意的二进制数据（比如图片、视频、音轨、软件程序）。当然，主体中也可以包含文本。

简单的报文实例

图 1-8 显示了可能会作为某个简单事务的一部分发送的 HTTP 报文。浏览器请求资源 http://www.joes-hardware.com/tools.html。

在图 1-8 中，浏览器发送了一条 HTTP 请求报文。这条请求的起始行中有一个 GET 命令，且本地资源为 /tools.html。这条请求说明它使用的是 1.0 版的 HTTP 协议。请求报文没有主体，因为从服务器上 GET 一个简单的文档不需要请求数据。

服务器会回送一条 HTTP 响应报文。这条响应中包含了 HTTP 的版本号（HTTP/1.0）、一个成功状态码（200）、一个描述性的原因短语（OK），以及一块响应首部字段，在所有这些内容之后跟着包含了所请求文档的响应主体。`Content-Length` 首部说明了响应主体的长度，`Content-Type` 首部说明了文档的 MIME 类型。

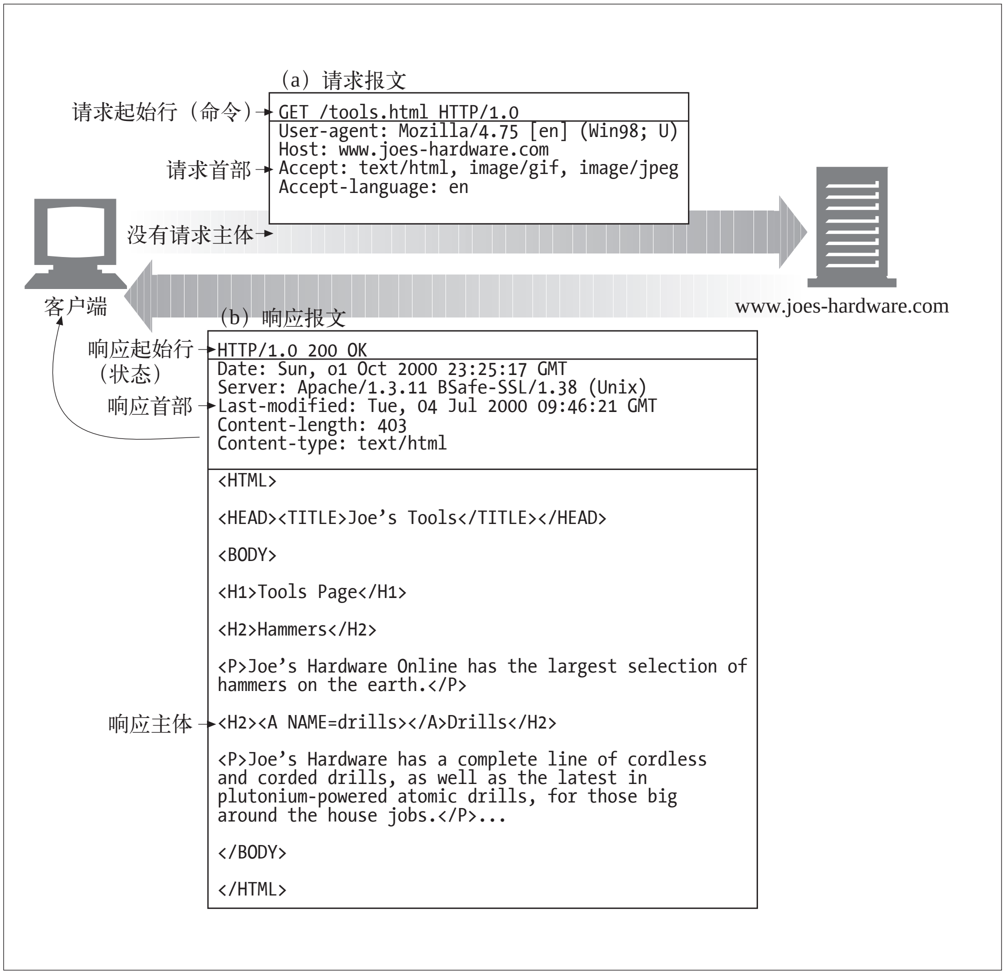


图 1-8　http://www.joes-hardware.com/tools.html 的 GET 事务实例

1.6　连接

概要介绍了 HTTP 报文的构成之后，我们来讨论一下报文是如何通过传输控制协议（Transmission Control Protocol，TCP）连接从一个地方搬移到另一个地方去的。

1.6.1　TCP/IP

HTTP 是个应用层协议。HTTP 无需操心网络通信的具体细节；它把联网的细节都交给了通用、可靠的因特网传输协议 TCP/IP。 11

TCP 提供了：

- 无差错的数据传输；
- 按序传输（数据总是会按照发送的顺序到达）；
- 未分段的数据流（可以在任意时刻以任意尺寸将数据发送出去）。

因特网自身就是基于 TCP/IP 的，TCP/IP 是全世界的计算机和网络设备常用的层次化分组交换网络协议集。TCP/IP 隐藏了各种网络和硬件的特点及弱点，使各种类型的计算机和网络都能够进行可靠地通信。

只要建立了 TCP 连接，客户端和服务器之间的报文交换就不会丢失、不会被破坏，也不会在接收时出现错序了。

用网络术语来说，HTTP 协议位于 TCP 的上层。HTTP 使用 TCP 来传输其报文数
12 据。与之类似，TCP 则位于 IP 的上层（参见图 1-9）。

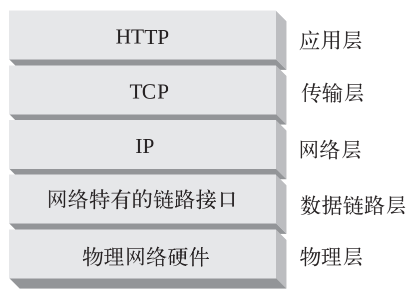


图 1-9　HTTP 网络协议栈

1.6.2　连接、IP地址及端口号

在 HTTP 客户端向服务器发送报文之前，需要用网际协议（Internet Protocol，IP）地址和端口号在客户端和服务器之间建立一条 TCP/IP 连接。

建立一条 TCP 连接的过程与给公司办公室的某个人打电话的过程类似。首先，要拨打公司的电话号码。这样就能进入正确的机构了。其次，拨打要联系的那个人的分机号。

在 TCP 中，你需要知道服务器的 IP 地址，以及与服务器上运行的特定软件相关的 TCP 端口号。

这就行了，但最初怎么获得 HTTP 服务器的 IP 地址和端口号呢？当然是通过 URL 了！我们前面曾提到过，URL 就是资源的地址，所以自然能够为我们提供存储资源的机器的 IP 地址。我们来看几个 URL：

    http://207.200.83.29:80/index.html
    http://www.netscape.com:80/index.html
    http://www.netscape.com/index.html

第一个 URL 使用了机器的 IP 地址，207.200.83.29 以及端口号 80。

第二个 URL 没有使用数字形式的 IP 地址，它使用的是文本形式的域名，或者称为主机名（www.netscape.com）。主机名就是 IP 地址比较人性化的别称。可以通过一种称为域名服务（Domain Name Service，DNS）的机制方便地将主机名转换为 IP 地址，这样所有问题就都解决了。第 2 章会介绍更多有关 DNS 和 URL 的内容。

最后一个 URL 没有端口号。HTTP 的 URL 中没有端口号时，可以假设默认端口号是 80。

有了 IP 地址和端口号，客户端就可以很方便地通过 TCP/IP 进行通信了。图 1-10 显示了浏览器是怎样通过 HTTP 显示位于远端服务器中的某个简单 HTML 资源的。 13

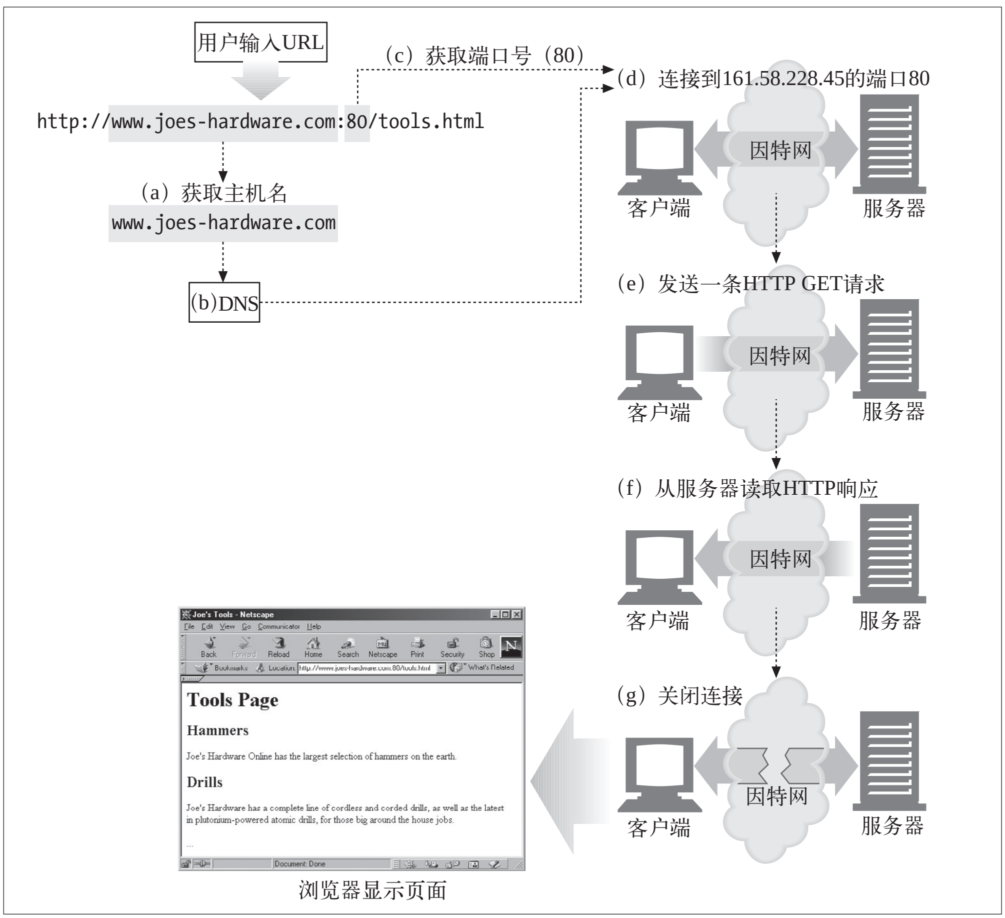


图 1-10　基本的浏览器连接处理 14

步骤如下：

(a) 浏览器从 URL 中解析出服务器的主机名；
(b) 浏览器将服务器的主机名转换成服务器的 IP 地址；
(c) 浏览器将端口号（如果有的话）从 URL 中解析出来；
(d) 浏览器建立一条与 Web 服务器的 TCP 连接；
(e) 浏览器向服务器发送一条 HTTP 请求报文；
(f) 服务器向浏览器回送一条 HTTP 响应报文；
(g) 关闭连接，浏览器显示文档。

1.6.3 一个使用Telnet的实例

由于 HTTP 使用了 TCP/IP 传输协议，而且它是基于文本的，没有使用那些难以理解的二进制格式，因此很容易直接与 Web 服务器进行对话。

Telnet 程序可以将键盘连接到某个目标 TCP 端口，并将此 TCP 端口的输出回送到显示屏上。Telnet 常用于远程终端会话，但它几乎可以连接所有的 TCP 服务器，包括 HTTP 服务器。

可以通过 Telnet 程序直接与 Web 服务器进行对话。通过 Telnet 可以打开一条到某台机器上某个端口的 TCP 连接，然后直接向那个端口输入一些字符。Web 服务器会将 Telnet 程序作为一个 Web 客户端来处理，所有回送给 TCP 连接的数据都会显示在屏幕上。

我们用 Telnet 与一个实际的 Web 服务器进行交互。我们要用 Telnet 获取 URL http://www.joes-hardware.com:80/tools.html 所指向的文档（你可以自己尝试一下这个实例）。

我们来看看会发生什么情况。

- 首先，查找 www.joes-hardware.com 的 IP 地址，打开一条到那台机器端口 80 的 TCP 连接。Telnet 会为我们完成那些“跑腿儿”的工作。
- 一旦打开了 TCP 连接，就要输入 HTTP 请求了。
- 请求结束（由一个空行表示）之后，服务器会在一条 HTTP 响应中将内容回送并关闭连接。

例 1-1 显示了对 http://www.joes-hardware.com:80/tools.html 的 HTTP 请求实例。我们输入的内容用粗体字表示。

例 1-1　一个使用 Telnet 的 HTTP 事务

```
% telnet www.joes-hardware.com 80
Trying 161.58.228.45...
Connected to joes-hardware.com.
Escape character is '^]'.
GET /tools.html HTTP/1.1
Host: www.joes-hardware.com

HTTP/1.1 200 OK
Date: Sun, 01 Oct 2000 23:25:17 GMT
Server: Apache/1.3.11 BSafe-SSL/1.38 (Unix) FrontPage/4.0.4.3
Last-Modified: Tue, 04 Jul 2000 09:46:21 GMT
ETag: "373979-193-3961b26d"
Accept-Ranges: bytes
Content-Length: 403
Connection: close
Content-Type: text/html

<HTML>
<HEAD><TITLE>Joe's Tools</TITLE></HEAD>
<BODY>
<H1>Tools Page</H1>
<H2>Hammers</H2>
<P>Joe's Hardware Online has the largest selection of hammers on the
earth.</P>
<H2><A NAME=drills></A>Drills</H2>
<P>Joe's Hardware has a complete line of cordless and corded drills,
as well as the latest
in plutonium-powered atomic drills, for those big around the house
jobs.</P> ...
</BODY>
</HTML>
Connection closed by foreign host.
```

15

Telnet 会查找主机名并打开一条连接，连接到在 www.joes-hardware.com 的端口 80 上监听的 Web 服务器。这条命令之后的三行内容是 Telnet 的输出，告诉我们它已经建立了连接。

然后我们输入最基本的请求命令 `GET/tools.html HTTP/1.1`，发送一个提供了源端主机名的 `Host` 首部，后面跟上一个空行，请求从服务器 www.joes-hardware.com 上获取资源 tools.html。随后，服务器会以一个响应行、几个响应首部、一个空行和最后面的 HTML 文档主体来应答。

要明确的是，Telnet 可以很好地模拟 HTTP 客户端，但不能作为服务器使用。而且对 Telnet 做脚本自动化是很繁琐乏味的。如果想要更灵活的工具，可以去看看 `nc`（netcat）。通过 `nc` 可以很方便地操纵基于 UDP 和 TCP 的流量（包括 HTTP），还可以为其编写脚本。更多细节参见 http://www.bgw.org/tutorials/utilities/nc.php[2]。

注 2：该链接已失效，读者可以访问 http://en.wikipedia.org/wiki/Netcat。（编者注）

1.7　协议版本

现在使用的 HTTP 协议有几个版本。HTTP 应用程序要尽量强健地处理各种不同的 HTTP 协议变体。目前仍在使用的版本如下。

- HTTP/0.9

  HTTP 的 1991 原型版本称为 HTTP/0.9。这个协议有很多严重的设计缺陷，只应该用于与老客户端的交互。HTTP/0.9 只支持 GET 方法，不支持多媒体内容的 MIME 类型、各种 HTTP 首部，或者版本号。HTTP/0.9 定义的初衷是为了获取简单的 HTML 对象，它很快就被 HTTP/1.0 取代了。

- HTTP/1.0

  1.0 是第一个得到广泛使用的 HTTP 版本。HTTP/1.0 添加了版本号、各种 HTTP 首部、一些额外的方法，以及对多媒体对象的处理。HTTP/1.0 使得包含生动图
16 片的 Web 页面和交互式表格成为可能，而这些页面和表格促使万维网为人们广泛地接受。这个规范从未得到良好地说明。在这个 HTTP 协议的商业演进和学术研究都在快速进行的时代，它集合了一系列的最佳实践。

- HTTP/1.0+

  在 20 世纪 90 年代中叶，很多流行的 Web 客户端和服务器都在飞快地向 HTTP 中添加各种特性，以满足快速扩张且在商业上十分成功的万维网的需要。其中很多特性，包括持久的 keep-alive 连接、虚拟主机支持，以及代理连接支持都被加入到 HTTP 之中，并成为非官方的事实标准。这种非正式的 HTTP 扩展版本通常称为 HTTP/1.0+。

- HTTP/1.1

  HTTP/1.1 重点关注的是校正 HTTP 设计中的结构性缺陷，明确语义，引入重要的性能优化措施，并删除一些不好的特性。HTTP/1.1 还包含了对 20 世纪 90 年代末正在发展中的更复杂的 Web 应用程序和部署方式的支持。HTTP/1.1 是当前使用的 HTTP 版本。

- HTTP-NG（又名 HTTP/2.0）

  HTTP-NG 是 HTTP/1.1 后继结构的原型建议，它重点关注的是性能的大幅优化，以及更强大的服务逻辑远程执行框架。HTTP-NG 的研究工作终止于 1998 年，编写本书时，还没有任何要用此建议取代 HTTP/1.1 的推广计划。更多信息请参见第 10 章。

1.8 Web的结构组件

在本章的概述中，我们重点介绍了两个 Web 应用程序（Web 浏览器和 Web 服务器）是如何相互发送报文来实现基本事务处理的。在因特网上，要与很多 Web 应用程序进行交互。在本节中，我们将列出其他一些比较重要的应用程序，如下所示。

- 代理

  位于客户端和服务器之间的 HTTP 中间实体。

- 缓存

  HTTP 的仓库，使常用页面的副本可以保存在离客户端更近的地方。

- 网关

  连接其他应用程序的特殊 Web 服务器。

- 隧道

  对 HTTP 通信报文进行盲转发的特殊代理。

- Agent 代理

  发起自动 HTTP 请求的半智能 Web 客户端。 17

1.8.1 代理

首先我们来看看 HTTP 代理服务器，这是 Web 安全、应用集成以及性能优化的重要组成模块。

如图 1-11 所示，代理位于客户端和服务器之间，接收所有客户端的 HTTP 请求，并将这些请求转发给服务器（可能会对请求进行修改之后转发）。对用户来说，这些应用程序就是一个代理，代表用户访问服务器。

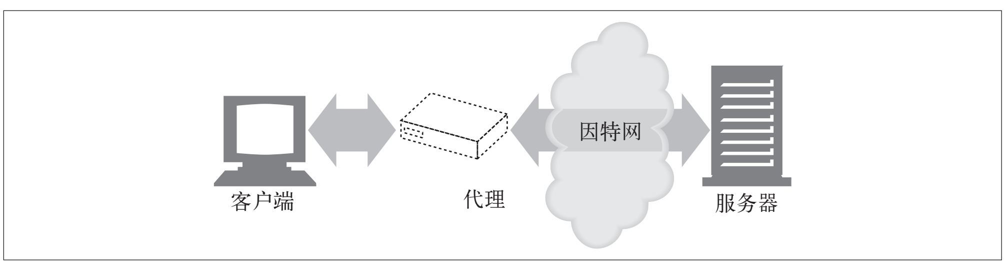
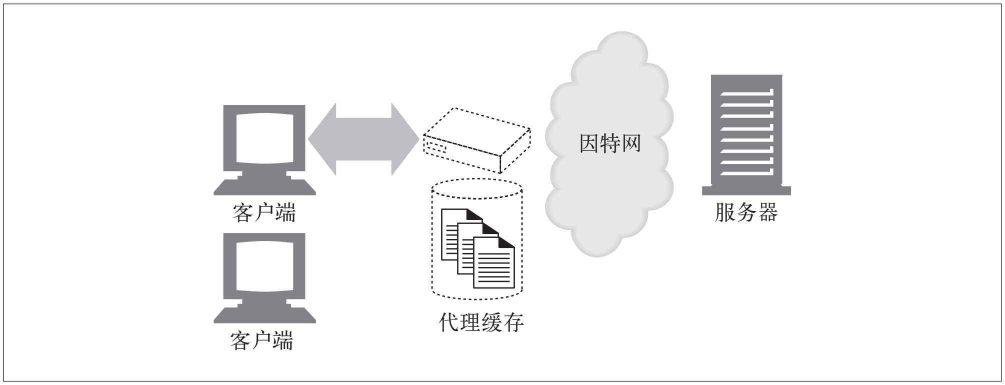


图 1-11　在客户端和服务器之间转发流量的代理

出于安全考虑，通常会将代理作为转发所有 Web 流量的可信任中间节点使用。代理还可以对请求和响应进行过滤。比如，在企业中对下载的应用程序进行病毒检测，或者对小学生屏蔽一些成人才能看的内容。我们将在第 6 章详细介绍代理。

1.8.2　缓存

Web 缓存（Web cache）或代理缓存（proxy cache）是一种特殊的 HTTP 代理服务器，可以将经过代理传送的常用文档复制保存起来。下一个请求同一文档的客户端就可以享受缓存的私有副本所提供的服务了（参见图 1-12）。



18 图 1-12　保存常用文档本地副本以提高性能的代理缓存

客户端从附近的缓存下载文档会比从远程 Web 服务器下载快得多。HTTP 定义了很多功能，使得缓存更加高效，并规范了文档的新鲜度和缓存内容的隐私性。第 7 章介绍了缓存技术。

1.8.3　网关

网关（gateway）是一种特殊的服务器，作为其他服务器的中间实体使用。通常用于将 HTTP 流量转换成其他的协议。网关接受请求时就好像自己是资源的源端服务器一样。客户端可能并不知道自己正在与一个网关进行通信。

例如，一个 HTTP/FTP 网关会通过 HTTP 请求接收对 FTP URI 的请求，但通过 FTP 协议来获取文档（参见图 1-13）。得到的文档会被封装成一条 HTTP 报文，发送给客户端。第 8 章将探讨网关。

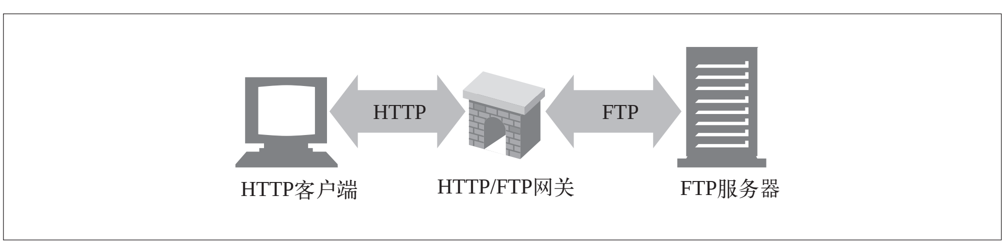


图 1-13　HTTP/FTP 网关

1.8.4 隧道

隧道（tunnel）是建立起来之后，就会在两条连接之间对原始数据进行盲转发的 HTTP 应用程序。HTTP 隧道通常用来在一条或多条 HTTP 连接上转发非 HTTP 数据，转发时不会窥探数据。

HTTP 隧道的一种常见用途是通过 HTTP 连接承载加密的安全套接字层（SSL，Secure Sockets Layer）流量，这样 SSL 流量就可以穿过只允许 Web 流量通过的防火墙了。如图 1-14 所示，HTTP/SSL 隧道收到一条 HTTP 请求，要求建立一条到目的地址和端口的输出连接，然后在 HTTP 信道上通过隧道传输加密的 SSL 流量，这样就可以将其盲转发到目的服务器上去了。

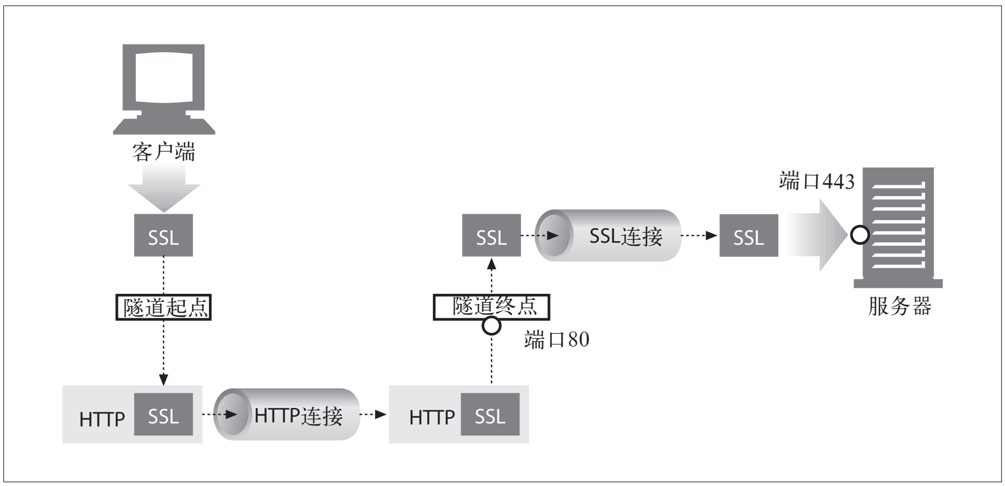
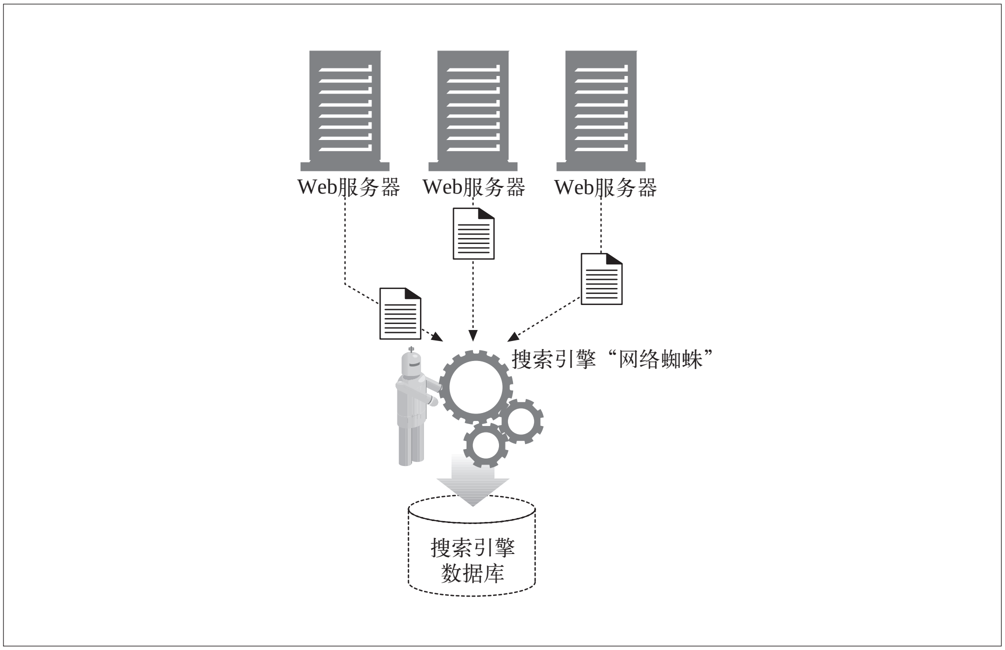


图 1-14　隧道可以在非 HTTP 网络上转发数据（显示的是 HTTP/SSL 隧道）

1.8.5 Agent代理

用户 Agent 代理（或者简称为 Agent 代理）是代表用户发起 HTTP 请求的客户端程序。所有发布 Web 请求的应用程序都是 HTTP Agent 代理。到目前为止，我们只提到过一种 HTTP Agent 代理：Web 浏览器，但用户 Agent 代理还有很多其他类型。 19

比如，有些自己会在 Web 上闲逛的自动用户 Agent 代理，可以在无人监视的情况下发布 HTTP 事务并获取内容。这些自动代理的名字通常都很生动，比如“网络蜘蛛”（spiders）或者“Web 机器人”（Web robots）（参见图 1-15）。网络蜘蛛会在 Web 上闲逛，搜集信息以构建有效的 Web 内容档案，比如一个搜索引擎的数据库或者为比较购物机器人生成的产品目录。更多信息请参见第 9 章。



20 图 1-15　自动搜索引擎“网络蜘蛛”就是 Agent 代理，可以从世界范围内获取 Web 页面

1.9　起始部分的结束语

这就是我们对 HTTP 的简要介绍。本章中，我们重点介绍了作为多媒体传输协议使用的 HTTP。概要说明了 HTTP 是怎样使用 URI 来命名远程服务器上的多媒体资源的，粗略介绍了如何利用 HTTP 请求和响应报文操纵远程服务器上的多媒体资源，最后考察了几种使用 HTTP 的 Web 应用程序。

本书的其余章节会更加详细地介绍 HTTP 协议、应用程序及资源的技术机制。

1.10　更多信息

本书稍后的章节将更深入地研究 HTTP，下面这些资源中也包含了与本章所涵盖的特定主题有关的背景知识。

1.10.1　HTTP协议信息

- *HTTP Pocket Reference*（《HTTP 口袋书》）
  Clinton Wong 著，O’Reilly & Associates 出版公司。这本书详细介绍了 HTTP，可以作为构成 HTTP 事务的首部和状态码的快速参考手册。

- http://www.w3.org/Protocols/
  这个 W3C 的 Web 页面中包含了很多与 HTTP 协议有关的重要链接。

- http://www.ietf.org/rfc/rfc2616.txt
  RFC2616“超文本传输协议——HTTP/1.1”是当前 HTTP 协议版本 HTTP/1.1 的官方规范。这个规范是一本编写流畅、组织良好而且非常详细的 HTTP 参考手册，但并不适于那些希望了解 HTTP 底层概念和动因，或者原理与实际应用之间区别的读者阅读。希望本书能够对这些底层概念进行补充，以便读者更好地使用这个规范。

- http://www.ietf.org/rfc/rfc1945.txt
  RFC1945“超文本传输协议——HTTP/1.0”是一个描述了 HTTP 现代基础的知识性 RFC。它对编写此规范时，已得到官方认可且具有“最佳实践”的 Web 应用程序行为进行了详细描述，还讨论了一些虽被 HTTP/1.1 所摒弃，但在一些老旧的应用程序中仍在广泛使用的行为。

- http://www.w3.org/Protocols/HTTP/AsImplemented.html
  这个 Web 页面介绍了 1991 年的 HTTP/0.9 协议，这个协议只实现了 GET 请求，而且不包含内容类型。 21

1.10.2 历史透视

- http://www.w3.org/Protocols/WhyHTTP.html
  这个简要的 Web 页面从 1991 年开始，从 HTTP 作者的角度，介绍了 HTTP 的一些起源以及初级目标。

- http://www.w3.org/History.html
  “A Little History of the World Wide Web”（万维网的简要历史）对万维网和 HTTP 的一些早期目标和构建基础进行了简短但有趣的剖析。

- http://www.w3.org/DesignIssues/Architecture.html
  “Web Architecture from 50,000 feet”（高空俯瞰 Web 结构）绘制了一幅广阔、远大的万维网蓝图，并详述了影响 HTTP 和相关 Web 技术的设计原则。

1.10.3 其他万维网信息

- http://www.w3.org
  W3C 是 Web 的科技驱动团队。W3C 致力于促进 Web 演化的互操作性技术（规范、准则、软件及工具）研究。W3C 站点是一个包含了 Web 技术简介和详细文档的宝库。

- http://www.ietf.org/rfc/rfc2396.txt
  RFC 2396“Uniform Resource Identifiers（URI）：Generic Syntax”，（“统一资源标识符（URI）：通用语法”）是 URI 和 URL 的详细参考。

- http://www.ieft.ofg/rfc/rfc2141.txt
  RFC2141“URN Syntax”（“URN 的语法”）是一个写于 1997 年的描述 URN 语法的规范。

- http://www.ietf.org/rfc/rfc2046.txt
  RFC2046“MIME Part 2：Media Types”（“MIME 第 II 部分：媒体类型”）是为进行多媒体内容管理而定义的多用途因特网邮件扩展标准的五部因特网规范中的第二部。

- http://www.wrec.org/Drafts/draft-ietf-wrec-taxonomy-06.txt
  这个因特网草案“Internet Web Replication and Caching Taxonomy”（“因特网
  22 Web 复制和缓存分类法”）解释了 Web 结构组件中的标准术语。

第2章

URL与资源

我们可以把因特网当作一个巨大的正在扩张的城市，里面充满了各种可看的东西，可做的事情。你和其他居民，以及到这个正在蓬勃发展的社区旅游的游客都要为这个城市大量的景点和服务使用标准命名规范。博物馆、饭店和家庭住址要使用街道地址，消防局、老板的秘书，以及抱怨你太少打电话给她的母亲要使用电话号码。

所有的东西都有一个标准化的名字，以帮助人们寻找城市中的各种资源。书籍有 ISBN 号，公交车有线路号，银行账户有账户编码，个人有社会保险号码。明天，你要到机场的 31 号出口去接你的生意伙伴。每天早上你都要乘坐红线火车，并在 Kendall 广场站出站。

所有人都对这些名字的标准达成了一致，所以才能方便地共享这座城市的宝藏。你告诉出租车司机把你载到 McAllister 大街 246 号，他就知道你是什么意思了（即使他走的是一条很远的路）。

URL 就是因特网资源的标准化名称。URL 指向一条电子信息片段，告诉你它们位于何处，以及如何与之进行交互。

本章，我们将介绍以下内容：

- URL 语法，以及各种 URL 组件的含义及其所做的工作；
- 很多 Web 客户端都支持的 URL 快捷方式，包括相对 URL 和自动扩展 URL；
- URL 编码和字符规则；
- 支持各种因特网信息系统的常见 URL 方案；
- URL 的未来，包括 URN——这种框架可以在对象从一处搬移到另一处时，保持
23 稳定的访问名称。

2.1 浏览因特网资源

URL 是浏览器寻找信息时所需的资源位置。通过 URL，人类和应用程序才能找到、使用并共享因特网上大量的数据资源。URL 是人们对 HTTP 和其他协议的常用访问点：一个人将浏览器指向一个 URL，浏览器就会在幕后发送适当的协议报文来获取人们所期望的资源。

URI 是一类更通用的资源标识符，URL 实际上是它的一个子集。URI 是一个通用的概念，由两个主要的子集 URL 和 URN 构成，URL 是通过描述资源的位置来标识资源的，而 URN（本章稍后会介绍）则是通过名字来识别资源的，与它们当前所处位置无关。

HTTP 规范将更通用的概念 URI 作为其资源标识符，但实际上，HTTP 应用程序处理的只是 URI 的 URL 子集。本书有时会不加区分地使用 URI 和 URL，但我们讲的基本上都是 URL。

比如说，你想要获取 URL http://www.joes-hardware.com/seasonal/index-fall.html。那么 URL 分以下三部分。

- URL 的第一部分（http）是 URL 方案（scheme）。方案可以告知 Web 客户端怎样访问资源。在这个例子中，URL 说明要使用 HTTP 协议。
- URL 的第二部分（www.joes-hardware.com）指的是服务器的位置。这部分告知 Web 客户端资源位于何处。
- URL 的第三部分（/seasonal/index-fall.html）是资源路径。路径说明了请求的是服务器上哪个特定的本地资源。

对此的说明请参见图 2-1。

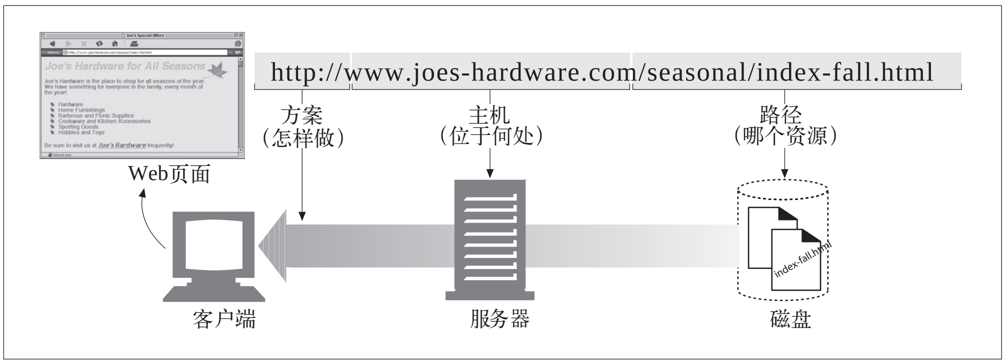


图 2-1　URL 是怎样与浏览器、客户端、服务器以及服务器文件系统中的位置进行关联的

URL 可以通过 HTTP 之外的其他协议来访问资源。它们可以指向因特网上的任意资源，或者个人的 E-mail 账户：

mailto:president@whitehouse.gov 24

或者通过其他协议（比如 FTP 协议）访问的各种文件：

ftp://ftp.lots-o-books.com/pub/complete-price-list.xls

或者从流视频服务器上下载电影：

rtsp://www.joes-hardware.com:554/interview/cto_video

URL 提供了一种统一的资源命名方式。大多数 URL 都有同样的：“方案 :// 服务器位置 / 路径”结构。因此，对网络上的每个资源以及获取这些资源的每种方式来说，命名资源的方法都只有一种，这样不管是谁都可以用名字来找到这个资源了。但是，事情并不是一开始就是这样的。

URL出现之前的黑暗岁月

在 Web 和 URL 出现之前，人们要靠分类杂乱的应用程序来访问分布在网络中的数据。大多数人都不会幸运地拥有所有合适的应用程序，或者不能够理解，也没有足够的耐心来使用这些程序。

在 URL 出现之前，要想和朋友共享 complete-catalog.xls 文件，就得说这样一些话："用 FTP 连接到 ftp.joes-hardware.com 上。用匿名登录，然后输入你的用户名作为密码。变换到 pub 目录。转换为二进制模式。现在，可以将名为 complete-catalog.xls 的文件下载到本地文件系统，并在那里浏览这个文件了。"

现在，像网景的 Navigator 和微软的 Internet Explorer 这样的浏览器都将很多这样的功能捆绑成一个便捷包。通过 URL，这些应用程序就可以通过一个接口，以统一的方式去访问许多资源了。只要说"将浏览器指向 ftp://ftp.lots-o-books.com/pub/complete-catalog.xls"就可以取代上面那些复杂的指令了。

URL 为应用程序提供了一种访问资源的手段。实际上，很多用户可能都不知道他们的浏览器在获取所请求资源时所使用的协议和访问方法。

有了 Web 浏览器，就不再需要用新闻阅读器来阅读因特网新闻，或者用 FTP 客户端来访问 FTP 服务器上的文件了，而且也无需用电子邮件程序来收发 E-mail 报文了。URL 告知浏览器如何对资源进行访问和处理，这有助于简化复杂的网络世界。[1] 应用程序可以使用 URL 来简化信息的访问过程。

URL 为用户及他们的浏览器提供了找到信息所需的所有条件。URL 定义了用户所需
25 的特定资源，它位于何处以及如何获取它。

2.2 URL的语法

URL 提供了一种定位因特网上任意资源的手段，但这些资源是可以通过各种不同的方案（比如 HTTP、FTP、SMTP）来访问的，因此 URL 语法会随方案的不同而有所不同。

这是不是意味着每种不同的 URL 方案都会有完全不同的语法呢？实际上，不是的。大部分 URL 都遵循通用的 URL 语法，而且不同 URL 方案的风格和语法都有不少重叠。

注 1：浏览器通常会用其他应用程序来处理特殊的资源。比如，Internet Explorer 就启动了一个 E-mail 应用程序来处理那些表示 E-mail 资源的 URL。

大多数 URL 方案的 URL 语法都建立在这个由 9 部分构成的通用格式上：

```
<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>
```

几乎没有哪个 URL 中包含了所有这些组件。URL 最重要的 3 个部分是方案（scheme）、主机（host）和路径（path）。表 2-1 对各种组件进行了总结。

表2-1　通用URL组件

组　件	描　述	默　认　值
方案	访问服务器以获取资源时要使用哪种协议	无
用户	某些方案访问资源时需要的用户名	匿名
密码	用户名后面可能要包含的密码，中间由冒号（:）分隔	<E-mail 地址 >
主机	资源宿主服务器的主机名或点分 IP 地址	无
端口	资源宿主服务器正在监听的端口号。很多方案都有默认端口号（HTTP 的默认端口号为 80）	每个方案特有
路径	服务器上资源的本地名，由一个斜杠（/）将其与前面的 URL 组件分隔开来。路径组件的语法是与服务器和方案有关的（本章稍后会讲到 URL 路径可以分为若干个段，每段都可以有其特有的组件。）	无
参数	某些方案会用这个组件来指定输入参数。参数为名 / 值对。URL 中可以包含多个参数字段，它们相互之间以及与路径的其余部分之间用分号（;）分隔	无
查询	某些方案会用这个组件传递参数以激活应用程序（比如数据库、公告板、搜索引擎以及其他因特网网关）。查询组件的内容没有通用格式。用字符“?”将其与 URL 的其余部分分隔开来	无
片段	一小片或一部分资源的名字。引用对象时，不会将 frag 字段传送给服务器；这个字段是在客户端内部使用的。通过字符“#”将其与 URL 的其余部分分隔开来	无

比如，我们来看看 URL：http://www.joes-hardware.com:80/index.html，其方案是 http，
主机为 www.joes-hardware.com，端口是 80，路径为 /index.html。 26

2.2.1　方案——使用什么协议

方案实际上是规定如何访问指定资源的主要标识符，它会告诉负责解析 URL 的应用程序应该使用什么协议。在我们这个简单的 HTTP URL 中所使用的方案就是 http。

方案组件必须以一个字母符号开始，由第一个“:”符号将其与 URL 的其余部分分隔开来。方案名是大小写无关的，因此 URL“http://www.joes-hardware.com”和“HTTP://www.joes-hardware.com”是等价的。

2.2.2　主机与端口

要想在因特网上找到资源，应用程序要知道是哪台机器装载了资源，以及在那台机器的什么地方可以找到能对目标资源进行访问的服务器。URL 的*主机*和*端口*组件提供了这两组信息。

主机组件标识了因特网上能够访问资源的宿主机器。可以用上述主机名（www.joes-hardware.com），或者 IP 地址来表示主机名。比如，下面两个 URL 就指向同一个资源——第一个 URL 是通过主机名，第二个是通过 IP 地址指向服务器的：

http://www.joes-hardware.com:80/index.html
http://161.58.228.45:80/index.html

端口组件标识了服务器正在监听的网络端口。对下层使用了 TCP 协议的 HTTP 来说，默认端口号为 80。

2.2.3　用户名和密码

更有趣的组件是*用户*和*密码*组件。很多服务器都要求输入用户名和密码才会允许用户访问数据。FTP 服务器就是这样一个常见的实例。这里有几个例子：

ftp://ftp.prep.ai.mit.edu/pub/gnu
ftp://anonymous@ftp.prep.ai.mit.edu/pub/gnu
ftp://anonymous:my_passwd@ftp.prep.ai.mit.edu/pub/gnu
http://joe:joespasswd@www.joes-hardware.com/sales_info.txt

第一个例子没有用户或密码组件，只有标准的方案、主机和路径。如果某应用程序使用的 URL 方案要求输入用户名和密码，比如 FTP，但用户没有提供，它通常会插入一个默认的用户名和密码。比如，如果向浏览器提供一个 FTP URL，但没
27 有指定用户名和密码，它就会插入 anonymous（匿名用户）作为你的用户名，并发送一个默认的密码（Internet Explorer 会发送 IEUser，Netscape Navigator 则会发送 mozilla）。

第二个例子显示了一个指定为 anonymous 的用户名。这个用户名与主机组件组合在一起，看起来就像 E-mail 地址一样。字符“@”将用户和密码组件与 URL 的其余部分分隔开来。

在第三个例子中，指定了用户名（anonymous）和密码（my_passwd），两者之间由字符“:”分隔。

2.2.4　路径

URL 的路径组件说明了资源位于服务器的什么地方。路径通常很像一个分级的文件系统路径。比如：

http://www.joes-hardware.com:80/seasonal/index-fall.html

这个 URL 中的路径为 /seasonal/index-fall.html，很像 UNIX 文件系统中的文件系统路径。路径是服务器定位资源时所需的信息。[2]可以用字符“/”将 HTTP URL 的路径组件划分成一些路径段（path segment）（还是与 UNIX 文件系统中的文件路径类似）。每个路径段都有自己的参数（param）组件。

2.2.5　参数

对很多方案来说，只有简单的主机名和到达对象的路径是不够的。除了服务器正在监听的端口，以及是否能够通过用户名和密码访问资源外，很多协议都还需要更多的信息才能工作。

负责解析 URL 的应用程序需要这些协议参数来访问资源。否则，另一端的服务器可能就不会为请求提供服务，或者更糟糕的是，提供错误的服务。比如，像 FTP 这样的协议，有两种传输模式，二进制和文本形式。你肯定不希望以文本形式来传送二进制图片，这样的话，二进制图片可能会变得一团糟。

为了向应用程序提供它们所需的输入参数，以便正确地与服务器进行交互，URL 中有一个参数组件。这个组件就是 URL 中的名值对列表，由字符“;”将其与 URL 的其余部分（以及各名值对）分隔开来。它们为应用程序提供了访问资源所需的所有附加信息。比如：

ftp://prep.ai.mit.edu/pub/gnu;type=d

在这个例子中，有一个参数 type=d，参数名为 type，值为 d。 28

如前所述，HTTP URL 的路径组件可以分成若干路径段。每段都可以有自己的参数。比如：

http://www.joes-hardware.com/hammers;sale=false/index.html;graphics=true

这个例子就有两个路径段，hammers 和 index.html。hammers 路径段有参数 sale，其值为 false。index.html 段有参数 graphics，其值为 true。

注 2：这是一种简化的说法。在 18.2 节我们会看到，路径并不总能为资源定位提供足够的信息。有时服务器还需要其他的信息。

2.2.6 查询字符串

很多资源，比如数据库服务，都是可以通过提问题或进行查询来缩小所请求资源类型范围的。

假设 Joe 的五金商店在数据库中维护着一个未售货物的清单，并可以对清单进行查询，以判断产品是否有货，那就可以用下列 URL 来查询 Web 数据库网关，看看编号为 12731 的条目是否有货：

http://www.joes-hardware.com/inventory-check.cgi?item=12731

这个 URL 的大部分都与我们见过的其他 URL 类似。只有问号（?）右边的内容是新出现的。这部分被称为*查询*（query）组件。URL 的查询组件和标识网关资源的 URL 路径组件一起被发送给网关资源。基本上可以将网关当作访问其他应用程序的访问点（第 8 章会对网关进行详细的讨论）。

图 2-2 中有一个作为 Joe 的五金商店清单查询应用程序的网关的服务器，在这个例子中向此服务器发送了一个查询组件。查询的目的是检查清单中是否有尺寸为 large、颜色为 blue 的条目 12731。

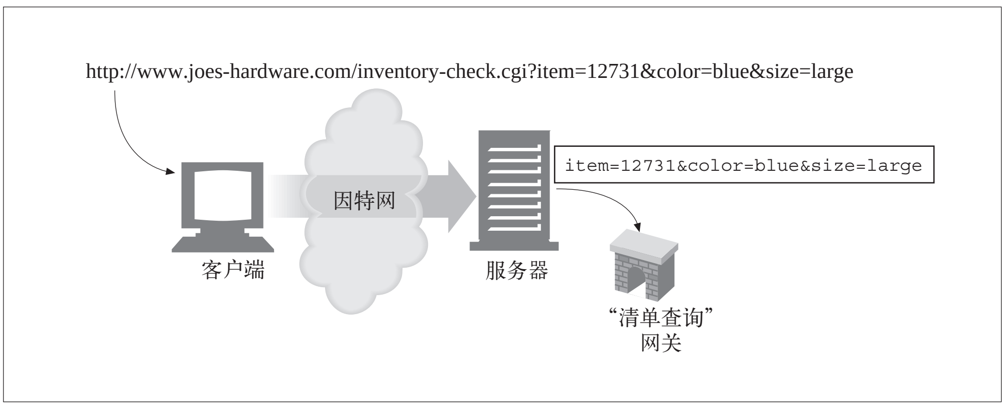


图 2-2 发送给网关应用程序的 URL 查询组件

在本章稍后会看到，除了有些不合规则的字符需要特别处理之外，对查询组件的格
29 式没什么要求。按照常规，很多网关都希望查询字符串以一系列"名 / 值"对的形式出现，名值对之间用字符"&"分隔：

http://www.joes-hardware.com/inventory-check.cgi?item=12731&color=blue

在这个例子中，查询组件有两个名 / 值对：item=12731 和 color=blue。

2.2.7 片段

有些资源类型，比如 HTML，除了资源级之外，还可以做进一步的划分。比如，对一个带有章节的大型文本文档来说，资源的 URL 会指向整个文本文档，但理想的情况是，能够指定资源中的那些章节。

为了引用部分资源或资源的一个片段，URL 支持使用*片段*（frag）组件来表示一个资源内部的片段。比如，URL 可以指向 HTML 文档中一个特定的图片或小节。

片段挂在 URL 的右手边，最前面有一个字符“#”。比如：

    http://www.joes-hardware.com/tools.html#drills

在这个例子中，片段 drills 引用了 Joe 的五金商店 Web 服务器上页面 /tools.html 中的一个部分。这部分的名字叫做 drills。

HTTP 服务器通常只处理整个对象，[3] 而不是对象的片段，客户端不能将片段传送给服务器（参见图 2-3）。浏览器从服务器获得了整个资源之后，会根据片段来显示你感兴趣的那部分资源。

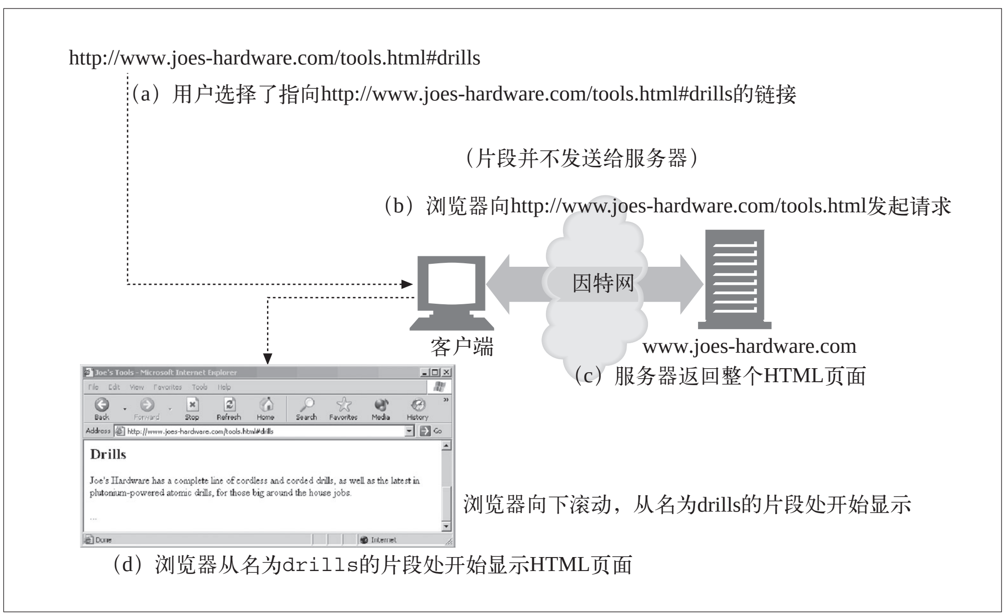


图 2-3　服务器处理的是整个对象，因此 URL 片段仅由客户端使用

注 3：在 15.9 节会看到 HTTP Agent 代理可能会请求某个字节范围内的对象，但在 URL 片段的上下文中，服务器会发送整个对象，由 Agent 代理将片段标识符应用于资源。

2.3 URL快捷方式

Web 客户端可以理解并使用几种 URL 快捷方式。相对 URL 是在某资源内部指定一个资源的便捷缩略方式。很多浏览器还支持 URL 的“自动扩展”，也就是用户输入 URL 的一个关键（可记忆的）部分，然后由浏览器将其余部分填充起来。2.3.2 节对此进行了解释。

2.3.1 相对URL

URL 有两种方式：*绝对的*和*相对的*。到目前为止，我们只见过绝对 URL。绝对
30 URL 中包含有访问资源所需的全部信息。

另一方面，相对 URL 是不完整的。要从相对 URL 中获取访问资源所需的全部信息，就必须相对于另一个，被称为其*基础*（base）的 URL 进行解析。

相对 URL 是 URL 的一种便捷缩略记法。如果你手工写过 HTML 的话，可能就会发现相对 URL 是多么便捷了。例 2-1 是一个嵌入了相对 URL 的 HTML 文档实例。

例 2-1　带有相对 URL 的 HTML 代码片段

```
<HTML>
<HEAD><TITLE>Joe's Tools</TITLE></HEAD>
<BODY>
<H1> Tools Page </H1>
<H2> Hammers <H2>
<P> Joe's Hardware Online has the largest selection of <A HREF="./
hammers.html">hammers
</A> on earth.
</BODY>
</HTML>
```

例 2-1 是资源：

http://www.joes-hardware.com/tools.html

的 HTML 文档。

在这个 HTML 文档中有一个包含了 URL ./hammers.html 的超链接。这个 URL 看起来是不完整的，但实际上是个合法的相对 URL。可以相对于它所在文档的 URL 对其进行解释；在这个例子中，就是相对于 Joe 的五金商店 Web 服务器的资源 /tools.
31 html。

使用缩略形式的相对 URL 语法，HTML 的编写者就可以省略 URL 中的方案、主机和其他一些组件了。这些组件可以从它们所属资源的*基础* URL 中推导出来。其他资源的 URL 也可以用这种缩略形式来表示。

在例 2-1 中，基础 URL 为：

http://www.joes-hardware.com/tools.html

用这个 URL 作为基础，可以推导出缺失的信息。我们知道资源名为 ./hammers.html，但并不知道方案或主机名是什么。通过这个基础 URL，可以推导出方案为 http，主机为 www.joes-hardware.com。图 2-4 对此进行了说明。



图 2-4　使用基础 URL

相对 URL 只是 URL 的片段或一小部分。处理 URL 的应用程序（比如浏览器）要能够在相对和绝对 URL 之间进行转换。

还需要注意的是，相对 URL 为保持一组资源（比如一些 HTML 页面）的可移植性提供了一种便捷方式。如果使用的是相对 URL，就可以在搬移一组文档的同时，仍然保持链接的有效性，因为相对 URL 都是相对于新基础进行解释的。这样就可以实现在其他服务器上提供镜像内容之类的功能了。

1. 基础URL

转换处理的第一步就是找到基础 URL。基础 URL 是作为相对 URL 的参考点使用的。可以来自以下几个不同的地方。

- *在资源中显式提供*
  有些资源会显式地指定基础 URL。比如，HTML 文档中可能会包含一个定义了基础 URL 的 HTML 标记 `<BASE>`，通过它来转换那个 HTML 文档中的所有相对 URL。

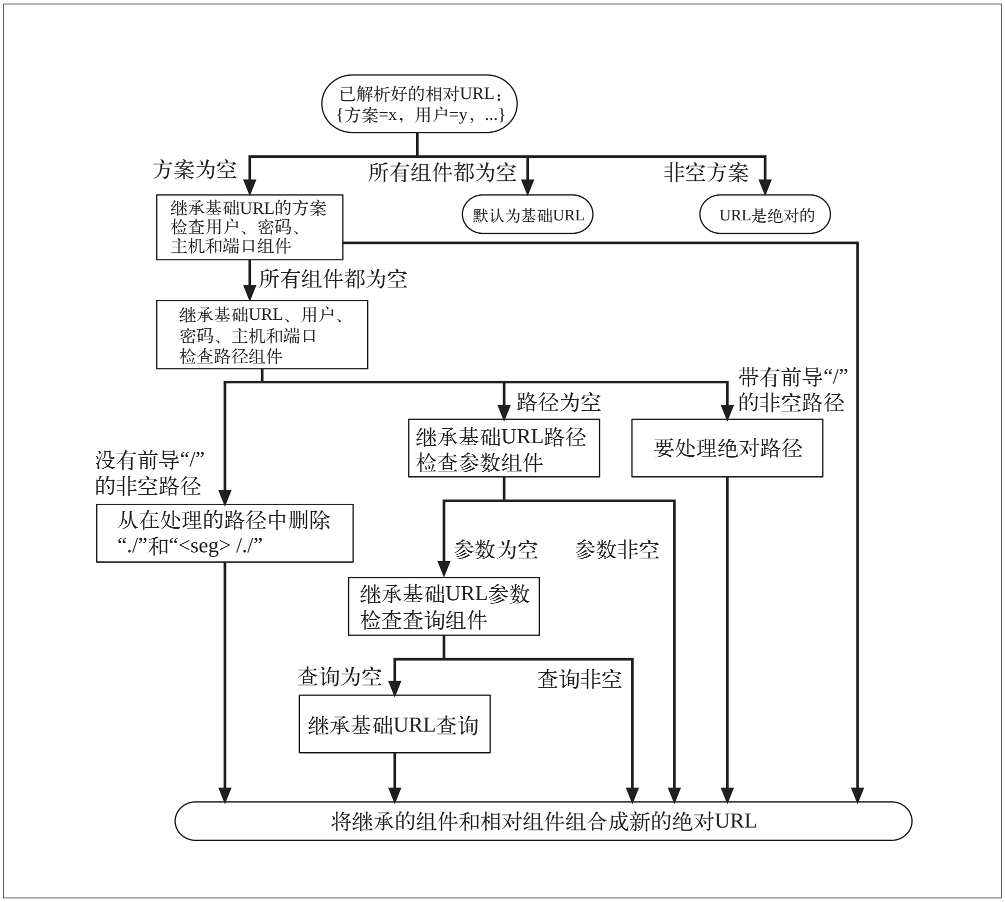
- *封装资源的基础 URL*
  如果在一个没有显式指定基础 URL 的资源中发现了一个相对 URL，如例 2-1 所示，可以将它所属资源的 URL 作为基础（如例中所示）。 32

- *没有基础 URL*
  在某些情况下，没有基础 URL。这通常意味着你有一个绝对 URL；但有时可能只是一个不完整或损坏了的 URL。

2. 解析相对引用

前面我们介绍了 URL 的基本组件和语法。要将相对 URL 转换为一个绝对 URL，下一步要做的就是将相对 URL 和基础 URL 划分成组件段。

实际上，这样只是在解析 URL，但这种做法会将其划分成一个个组件，因此通常会称作分解（decomposing）URL。只要将基础和相对 URL 划分成了组件，就可以应用图 2-5 中的算法来完成转换了。



33 图 2-5 将相对 URL 转换成绝对 URL

这个算法将一个相对 URL 转换成了其绝对模式，之后就可以用它来引用资源了。这个算法最初是在 RFC 1808 中制定的，后来被合并到了 RFC 2396 中。

可以对例 2-1 中的 ./hammers.html 实例使用图 2-5 中描述的算法。

(1) 路径为 ./hammers.html，基础 URL 为 http://www.joes-hardware.com/tools.html。
(2) 方案为空，沿着图表的左半边向下处理，继承基础 URL 方案（HTTP）。
(3) 至少一个组件非空，一直处理到底端，继承主机和端口组件。
(4) 将来自相对 URL（路径：./hammers.html）的组件与我们继承来的组件（方案：http，主机：www.joes-hardware.com，端口：80）合并起来，得到新的绝对 URL：http ://www.joes-hardware.com/hammers.html。

2.3.2 自动扩展URL

有些浏览器会在用户提交 URL 之后，或者在用户输入的时候尝试着自动扩展 URL。这就为用户提供了一条捷径：用户不需要输入完整的 URL，因为浏览器会自动扩展。

这些“自动扩展”特性有以下两种方式。

- *主机名扩展*

  在主机名扩展中，只要有些小提示，浏览器通常就可以在没有帮助的情况下，将你输入的主机名扩展为完整的主机名。

  比如，如果在地址栏中输入 yahoo，浏览器就会自动在主机名中插入 www. 和 .com，构建出 www.yahoo.com。如果找不到与 yahoo 匹配的站点，有些浏览器会在放弃之前尝试几种扩展形式。浏览器通过这些简单的技巧来节省你的时间，减少找不到的可能。

  但是，这些主机名扩展技巧可能会为其他一些 HTTP 应用程序带来问题，比如代理。第 6 章将详细讨论这些问题。

- *历史扩展*

  浏览器用来节省用户输入 URL 时间的另一种技巧是，将以前用户访问过的 URL 历史存储起来。当你输入 URL 时，它们就可以将你输入的 URL 与历史记录中 URL 的前缀进行匹配，并提供一些完整的选项供你选择。因此，如果你输入了一个以前访问过的 URL 的开始部分，比如 http://www.joes-，浏览器就可能会建议使用 http://www.joes-hardware.com。然后你就可以选择这个地址，不用输入完整的 URL 了。 34

注意，与代理共同使用时，URL 自动扩展的行为可能会有所不同。6.5.6 节将对此进行进一步讨论。

2.4 各种令人头疼的字符

URL 是*可移植的*（portable）。它要统一地命名因特网上所有的资源，这也就意味着要通过各种不同的协议来传送这些资源。这些协议在传输数据时都会使用不同的机制，所以，设计 URL，使其可以通过任意因特网协议安全地传输是很重要的。

安全传输意味着 URL 的传输不能丢失信息。有些协议，比如传输电子邮件的简单邮件传输协议（Simple Mail Transfer Protocol，SMTP），所使用的传输方法就会剥去一些特定的字符。[4] 为了避开这些问题，URL 只能使用一些相对较小的、通用的安全字母表中的字符。

除了希望 URL 可以被所有因特网协议进行传送之外，设计者们还希望 URL 也可供人类阅读。因此，即使不可见、不可打印的字符能够穿过邮件程序，从而成为可移植的，也不能在 URL 中使用。[5]

URL 还得是*完整的*，这就使问题变得更加复杂了。URL 的设计者们认识到有时人们可能会希望 URL 中包含除通用的安全字母表之外的二进制数据或字符。因此，需要有一种转义机制，能够将不安全的字符编码为安全字符，再进行传输。

本节总结了 URL 的通用字母表和编码规则。

2.4.1 URL字符集

默认的计算机系统字符集通常都倾向于以英语为中心。从历史上来看，很多计算机应用程序使用的都是 US-ASCII 字符集。US-ASCII 使用 7 位二进制码来表示英文打字机提供的大多数按键和少数用于文本格式和硬件通知的不可打印控制字符。

由于 US-ASCII 的历史悠久，所以其可移植性很好。但是，虽然美国用户使用起来很便捷，它却并不支持在各种欧洲语言或全世界数十亿人使用的数百种非罗马语言
35 中很常见的变体字符。

而且，有些 URL 中还会包含任意的二进制数据。认识到对完整性的需求之后，URL 的设计者就将*转义序列*集成了进去。通过转义序列，就可以用 US-ASCII 字符集的有限子集对任意字符值或数据进行编码了，这样就实现了可移植性和完整性。

2.4.2 编码机制

为了避开安全字符集表示法带来的限制，人们设计了一种编码机制，用来在 URL 中表示各种不安全的字符。这种编码机制就是通过一种“转义”表示法来表示不安全

注 4：这是由报文的 7 位二进制码编码造成的。如果源端是以 8 位或更多位编码的，就会有部分信息被剥离。
注 5：不可打印字符中包括空格符（注意，RFC 2396 建议应用程序忽略空格符）。

字符的，这种转义表示法包含一个百分号（%），后面跟着两个表示字符 ASCII 码的十六进制数。

表 2-2 中列出了几个例子。

表2-2　一些编码字符示例

字　符	ASCII码	示例URL
~	126(0x7E)	http://www.joes-hardware.com/%7Ejoe
空格	32(0x20)	http://www.joes-hardware.com/more%20tools.html
%	37(0x25)	http://www.joes-hardware.com/100%25satisfaction.html

2.4.3　字符限制

在 URL 中，有几个字符被保留起来，有着特殊的含义。有些字符不在定义的 US-ASCII 可打印字符集中。还有些字符会与某些因特网网关和协议产生混淆，因此不赞成使用。

表 2-3 列出了一些字符，在将其用于保留用途之外的场合时，要在 URL 中对其进行编码。

表2-3　保留及受限的字符

字　符	保留/受限
%	保留作为编码字符的转义标志
/	保留作为路径组件中分隔路径段的定界符
.	保留在路径组件中使用
..	保留在路径组件中使用
#	保留作为分段定界符使用
?	保留作为查询字符串定界符使用
;	保留作为参数定界符使用
:	保留作为方案、用户 / 口令，以及主机 / 端口组件的定界符使用
$, +	保留
@ & =	在某些方案的上下文中有特殊的含义，保留
{ } | \ ^ ~ [ ] '	由于各种传输 Agent 代理，比如各种网关的不安全处理，使用受限 36
< > "	不安全；这些字符在 URL 范围之外通常是有意义的，比如在文档中对 URL 自身进行定界（比如 http://www.joes-hardware.com），所以应该对其进行编码
0x00-0x1F, 0x7F	受限，这些十六进制范围内的字符都在 US-ASCII 字符集的不可打印区间内
>0x7F	受限，十六进制值在此范围内的字符都不在 US-ASCII 字符集的 7 比特范围内

2.4.4　另外一点说明

你可能会感到奇怪，为什么使用一些不安全字符的时候并没有发生什么不好的事情。比如，你可以访问 http://www.joes-hardware.com/~joe 上的 Joe 主页，而无需对“~”字符进行编码。对某些传输协议来说，这并不是什么问题，但对应用程序开发人员来说，对非安全字符进行编码仍然是明智的。

应用程序要按照一定规范工作。客户端应用程序在向其他应用程序发送任意 URL 之前，最好把所有不安全或受限字符都进行转换[6]。只要对所有不安全字符都进行了编码，这个 URL 就是可在各应用程序之间共享的*规范形式*；也就无需操心其他应用程序会被字符的任何特殊含义所迷惑了。

最适合判断是否需要对字符进行编码的程序就是从用户处获取 URL 的源端应用程序。URL 的每个组件都会有自己的安全 / 不安全字符，哪些字符是安全 / 不安全的与方案有关，因此只有从用户那里接收 URL 的应用程序才能够判断需要对哪些字符进行编码。

当然，另一种极端的做法就是应用程序对所有字符都进行编码。尽管并不建议这么做，但也没有什么强硬且严格的规则规定不能对那些安全字符进行编码；但在实际应用中，有些应用程序可能会假定不对安全字符进行编码，这么做的话可能会产生一些奇怪的破坏性行为。

有时，有些人会恶意地对额外的字符进行编码，以绕过那些对 URL 进行模式匹配的应用程序——比如，Web 过滤程序。对安全的 URL 组件进行编码会使模式匹配程序无法识别出它们所要搜寻的模式。总之，解释 URL 的应用程序必须在处理 URL
37 之前对其进行解码。

有些 URL 组件要便于识别，并且必须由字母开头，比如 URL 的方案。更多关于不同 URL 组件中保留字符和不安全字符的使用指南请回顾 2.2 节[7]。

2.5　方案的世界

本节将介绍更多 Web 常用方案格式。附录 A 给出了一个相当完整的方案列表，及各种方案文档的参考文献。

注 6：这里我们特指的是客户端应用程序，而不是其他的 HTTP 中间点，比如代理。6.5.5 节探讨了代理和其他中间 HTTP 应用程序试图代表客户端修改（比如编码）URL 时可能产生的一些问题。

注 7：表 2-3 列出了各种 URL 组件的保留字符。总之，只应该对这些在传输过程中不安全的字符进行编码。

表 2-4 总结了最常见的一些方案。回顾一下 2.2 节有助于理解表格中的语法部分。

表2-4　常见的方案格式

方　案	描　述
http	超文本传输协议方案，除了没有用户名和密码之外，与通用的 URL 格式相符。如果省略了端口，就默认为 80。 基本格式： http://<host>:<port>/<path>?<query>#<frag> 示例： http://www.joes-hardware.com/index.html http://www.joes-hardware.com:80/index.html
https	方案 https 与方案 http 是一对。唯一的区别在于方案 https 使用了网景的 SSL，SSL 为 HTTP 连接提供了端到端的加密机制。其语法与 HTTP 的语法相同，默认端口为 443。 基本格式： https://<host>:<port>/<path>?<query>#<frag> 示例： https://www.joes-hardware.com/secure.html
mailto	Mailto URL 指向的是 E-mail 地址。由于 E-mail 的行为与其他方案都有所不同（它并不指向任何可以直接访问的对象），所以 mailto URL 的格式与标准 URL 的格式也有所不同。因特网 E-mail 地址的语法记录在 RFC 822 中。 基本格式： mailto:<RFC-822-addr-spec> 示例： mailto:joe@joes-hardware.com 38
ftp	文件传输协议 URL 可以用来从 FTP 服务器上下载或向其上载文件，并获取 FTP 服务器上的目录结构内容的列表。 在 Web 和 URL 出现之前 FTP 就已经存在了。Web 应用程序将 FTP 作为一种数据访问方案使用。URL 语法遵循下列通用格式。 基本格式： ftp://<user>:<password>@<host>:<port>/<path>;<params> 示例： ftp://anonymous:joe%40joes-hardware.com@prep.ai.mit.edu:21/pub/gnu/
rtsp，rtspu	RTSP URL 是可以通过实时流传输协议（Real Time Streaming Protocol）解析的音 / 视频媒体资源的标识符。 方案 rtspu 中的 u 表示它是使用 UDP 协议来获取资源的。 基本格式： rtsp://<user>:<password>@<host>:<port>/<path> rtspu://<user>:<password>@<host>:<port>/<path> 示例： rtsp://www.joes-hardware.com:554/interview/cto_video

（续）

方　　案	描　　述
file	方案 file 表示一台指定主机（通过本地磁盘、网络文件系统或其他一些文件共享系统）上可直接访问的文件。各字段都遵循通用格式。如果省略了主机名，就默认为正在使用 URL 的本地主机。 基本格式： file://<host>/<path> 示例： file://OFFICE-FS/policies/casual-fridays.doc
news	根据 RFC 1036 的定义，方案 news 用来访问一些特定的文章或新闻组。它有一个很独特的性质：news URL 自身包含的信息不足以对资源进行定位。 news URL 中缺乏到何处获取资源的信息——没有提供主机名或机器名称。从用户那里获取此类信息是解释程序的工作。比如，在网景浏览器的“选项”（Options）菜单中，就可以指定自己的 NNTP（news）服务器。这样，浏览器有了 news URL 的时候就知道应该使用哪个服务器了。 新闻资源可以从多台服务器中获得。它们被称为位置无关的，因为对它们的访问不依赖于任何一个源服务器。 news URL 中保留了字符“@”，用来区分指向新闻组的 news URL 和指向特定新闻文章的 news URL。 基本格式： news:<newsgroup> news:<news-article-id> 示例： 39 news:rec.arts.startrek
telnet	方案 telnet 用于访问交互式业务。它表示的并不是对象自身，而是可通过 telnet 协议访问的交互式应用程序（资源）。 基本格式： telnet://<user>:<password>@<host>:<port>/ 示例： telnet://slurp:webhound@joes-hardware.com:23/

2.6　未来展望

URL 是一种强有力的工具。它可以用来命名所有现存对象，而且可以很方便地包含一些新格式。URL 还提供了一种可以在各种因特网协议间共享的统一命名机制。

但 URL 并不完美。它们表示的是实际的地址，而不是准确的名字。这就意味着 URL 会告诉你资源此时处于什么位置。它会为你提供特定端口上特定服务器的名字，告诉你在何处可以找到这个资源。这种方案的缺点在于如果资源被移走了，URL 也就不再有效了。那时，它就无法对对象进行定位了。

如果有了对象的准确名称，则不论其位于何处都可以找到这个对象，那该多完美啊。就像人一样，只要给定了资源的名称和其他一些情况，无论资源移到何处，你都能够追踪到它。

为了应对这个问题，因特网工程任务组（Internet Engineering Task Force，IETF）已经对一种名为统一资源名（uniform resource name，URN）的新标准做了一段时间的研究了。无论对象搬移到什么地方（在一个 Web 服务器内或是在不同的 Web 服务器间），URN 都能为对象提供一个稳定的名称。

永久统一资源定位符（persistent uniform resource locators，PURL）是用 URL 来实现 URN 功能的一个例子。其基本思想是在搜索资源的过程中引入另一个中间层，通过一个中间资源定位符（resource locator）服务器对资源的实际 URL 进行登记和跟踪。客户端可以向定位符请求一个永久 URL，定位符可以以一个资源作为响应，将客户端重定向到资源当前实际的 URL 上去（参见图 2-6）。更多有关 PURL 的信息，请访问 http://purl.oclc.org。

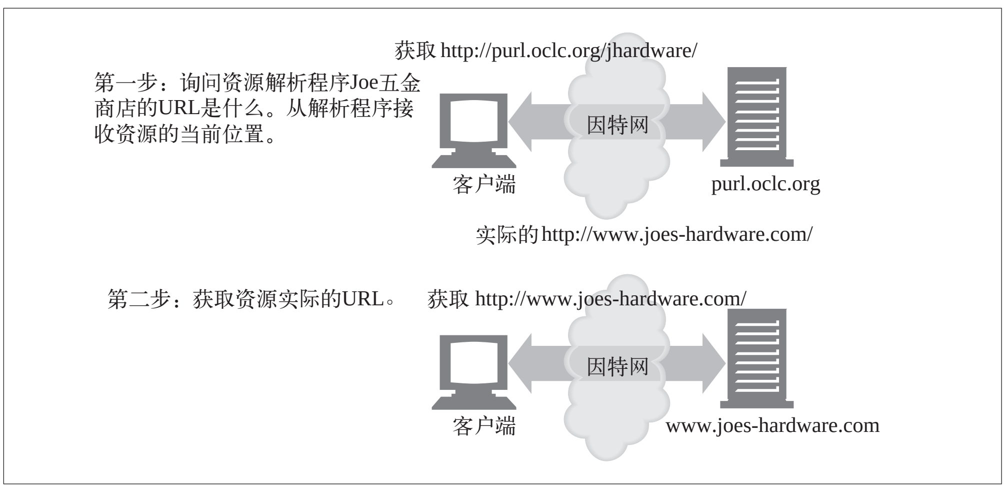


图 2-6　PURL 通过资源定位符服务器来命名资源的当前位置

如果不是现在，那是什么时候

URN 背后的思想已经提出一段时间了。实际上，如果去看看某些相关规范的发布日期，你可能会问，为什么它们现在都还没有投入使用。 40

从 URL 转换成 URN 是一项巨大的工程。标准化工作的进程很缓慢，而且通常都有很充分的理由。支持 URN 需要进行很多改动——标准主体的一致性，对各种 HTTP

应用程序的修改等。做这种改动需要进行大量的工作，而且很不幸（或者可能很幸运）的是 URL 还有很大的能量，还要等待更合适的时机才能进行这种转换。

在 Web 爆炸性增长的过程中，因特网用户——包括从计算机科学家到普通因特网用户的每一个人——都已经学会使用 URL 了。在备受笨拙语法（对新手来说）和顽固问题困扰的同时，人们已经学会了如何使用 URL，以及如何对付它们的一些缺陷。URL 有一些限制，但这并不是 Web 开发社区所面临的最紧迫的问题。

目前看来，在可预见的未来，因特网资源仍然会以 URL 来命名。它们无处不在，而且是 Web 的成功过程中一个非常重要的部分。其他命名方案想要取代 URL 还要一段时间。但是，URL 确实有其局限型，可能会出现新的标准（可能就是 URN），对这种标准进行部署会解决其中的某些问题。

2.7 更多信息

更多有关 URL 的信息，请参考以下资源。

- http://www.w3.org/Addressing/
  这是 W3C 有关 URI 和 URL 命名及寻址的 Web 页面。

- http://www.ietf.org/rfc/rfc1738
  RFC 1738，T. Berners-Lee、L. Masinter 和 M. McCahill 编写的“Uniform Resource
  41 Locators（URL）”（“统一资源定位符”）。

- http://www.ietf.org/rfc/rfc2396.txt
  RFC 2396，T. Berners-Lee、R. Fielding 和 L. Masinter 编写的“Uniform Resource Identifiers（URI）：Generic Syntax”（“URI：通用语法”）。

- http://www.ietf.org/rfc/rfc2141.txt
  RFC 2141，R. Moats 编写的“URN Syntax”（“URN 语法”）。

- http://purl.oclc.org
  永久统一资源定位符的 Web 站点。

- http://www.ietf.org/rfc/rfc1808.txt
  RFC 1808，R. Fielding 编写的“Relative Uniform Resource Locators”（“相对统
  42 一资源定位符”）。

第3章

HTTP报文

如果说 HTTP 是因特网的信使，那么 HTTP 报文就是它用来搬东西的包裹了。第 1 章说明了 HTTP 程序是怎样互相发送报文来完成工作的。本章则会介绍所有与 HTTP 报文有关的事情——如何创建报文，以及如何理解它们。通过阅读本章，就可以了解编写自己的 HTTP 应用程序所需掌握的大部分内容。具体来说，你会理解下列概念：

- 报文是如何流动的；
- HTTP 报文的三个组成部分（起始行、首部和实体的主体部分）；
- 请求和响应报文之间的区别；
- 请求报文支持的各种功能（方法）；
- 和响应报文一起返回的各种状态码；
- 各种各样的 HTTP 首部都是用来做什么的。

3.1　报文流

HTTP 报文是在 HTTP 应用程序之间发送的数据块。这些数据块以一些文本形式的*元信息*（meta-information）开头，这些信息描述了报文的内容及含义，后面跟着可选的数据部分。这些报文在客户端、服务器和代理之间流动。术语“流入”、“流出”、“上游”及“下游”都是用来描述报文方向的。

3.1.1　报文流入源端服务器

HTTP 使用术语*流入*（inbound）和*流出*（outbound）来描述*事务处理*（transaction）的方向。报文流入源端服务器，工作完成之后，会流回用户的 Agent 代理中（参见
43 图 3-1）。

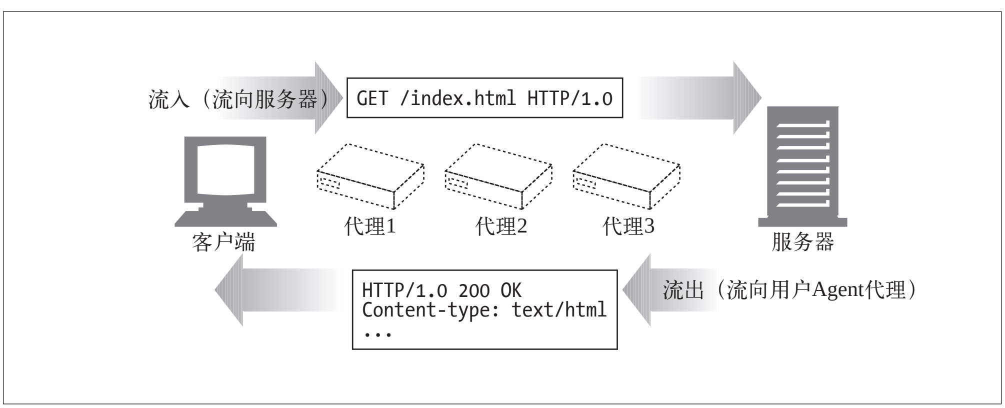


图 3-1　报文流入源端服务器并流回到客户端

3.1.2 报文向下游流动

HTTP 报文会像河水一样流动。不管是请求报文还是响应报文，所有报文都会向下游（downstream）流动（参见图 3-2）。所有报文的发送者都在接收者的上游（upstream）。在图 3-2 中，对请求报文来说，代理 1 位于代理 3 的上游，但对响应报文来说，它就位于代理 3 的下游[1]。

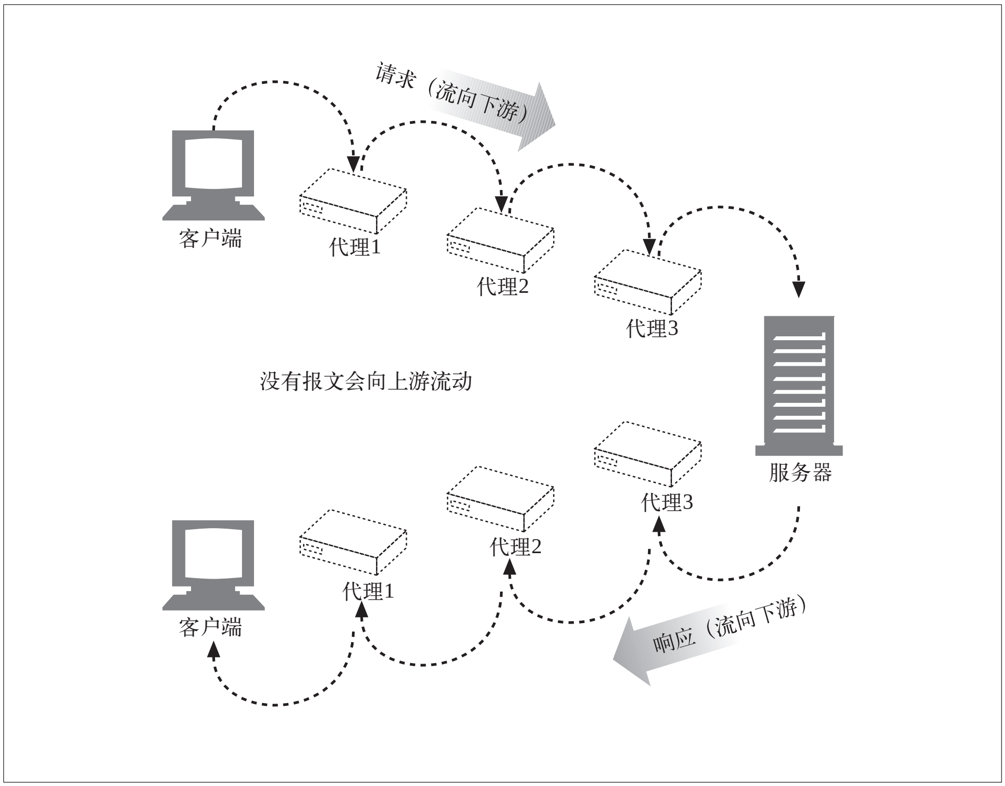


图 3-2 所有报文都向下游流动

3.2 报文的组成部分

HTTP 报文是简单的格式化数据块。看一下图 3-3 给出的例子。每条报文都包含一条来自客户端的请求，或者一条来自服务器的响应。它们由三个部分组成：对报文进行描述的*起始行*（start line）、包含属性的*首部*（header）块，以及可选的、包含数据的*主体*（body）部分。

注 1：术语“上游”和“下游”都只与发送者和接收者有关。我们无法区分报文是发送给源端服务器的还是发送给客户端的，因为两者都是下游节点。

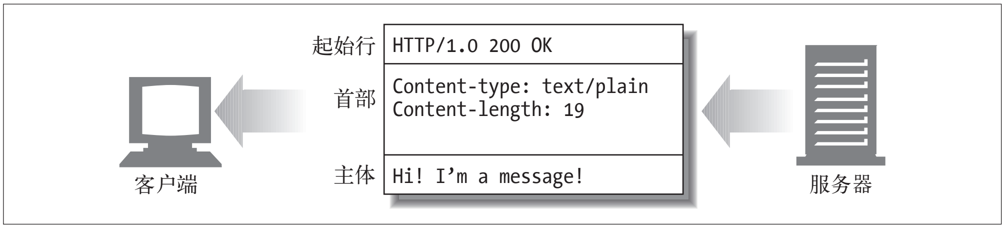


图 3-3 HTTP 报文的三个部分

起始行和首部就是由行分隔的 ASCII 文本。每行都以一个由两个字符组成的行终止序列作为结束，其中包括一个回车符（ASCII 码 13）和一个换行符（ASCII 码 10）。这个行终止序列可以写做 CRLF。需要指出的是，尽管 HTTP 规范中说明应该用 CRLF 来表示行终止，但稳健的应用程序也应该接受单个换行符作为行的终止。有些老的，或不完整的 HTTP 应用程序并不总是既发送回车符，又发送换行符。

实体的主体或报文的主体（或者就称为主体）是一个可选的数据块。与起始行和首部不同的是，主体中可以包含文本或二进制数据，也可以为空。

在图 3-3 的例子中，首部给出了一些与主体有关的信息。`Content-Type` 行说明了主体是什么——在这个例子中，就是纯文本文档。`Content-Length` 行说明了主体有多大，在这里就只有 19 个字节。

44

3.2.1 报文的语法

所有的 HTTP 报文都可以分为两类：*请求报文*（request message）和*响应报文*（response message）。请求报文会向 Web 服务器请求一个动作。响应报文会将请求的结果返回给客户端。请求和响应报文的基本报文结构相同。图 3-4 显示了获取一张 GIF 图片所需的请求和响应报文。

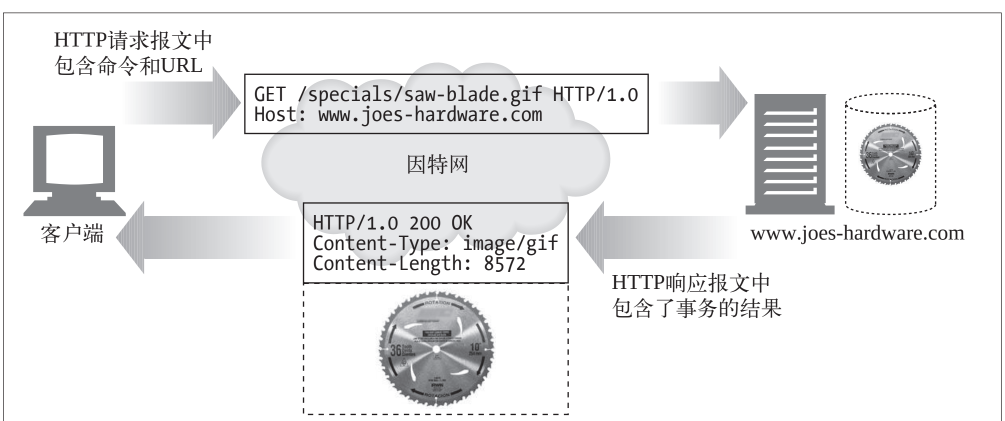


图 3-4 包含请求和响应报文的 HTTP 事务

这是请求报文的格式： 45

```
<method> <request-URL> <version>
<headers>

<entity-body>
```

这是响应报文的格式（注意，只有起始行的语法有所不同）：

```
<version> <status> <reason-phrase>
<headers>

<entity-body>
```

下面是对各部分的简要描述。

- *方法*（method）
  客户端希望服务器对资源执行的动作。是一个单独的词，比如 GET、HEAD 或 POST。本章稍后将详细介绍方法。

- *请求 URL*（request-URL）
  命名了所请求资源，或者 URL 路径组件的完整 URL。如果直接与服务器进行对话，只要 URL 的路径组件是资源的绝对路径，通常就不会有什么问题——服务器可以假定自己是 URL 的主机 / 端口。第 2 章详细地介绍了 URL 的语法。

- *版本*（version）
  报文所使用的 HTTP 版本，其格式看起来是这样的：

  ```
  HTTP/<major>.<minor>
  ```

  其中*主要版本号*（major）和*次要版本号*（minor）都是整数。本章稍后会详细说明 HTTP 的版本问题。

- *状态码*（status-code）
  这三位数字描述了请求过程中所发生的情况。每个状态码的第一位数字都用于描述状态的一般类别（“成功”、“出错”等）。本章稍后提供了 HTTP 规范定义的状态码及其含义的完整列表。 46

- *原因短语*（reason-phrase）
  数字状态码的可读版本，包含行终止序列之前的所有文本。本章稍后提供了 HTTP 规范定义的所有状态码的原因短语示例。原因短语只对人类有意义，因此，比如说，尽管响应行 HTTP/1.0 200 NOT OK 和 HTTP/1.0 200 OK 中原因短语的含义不同，但同样都会被当作成功指示处理。

- 首部（header）
  可以有零个或多个首部，每个首部都包含一个名字，后面跟着一个冒号（:），然后是一个可选的空格，接着是一个值，最后是一个 CRLF。首部是由一个空行（CRLF）结束的，表示了首部列表的结束和实体主体部分的开始。有些 HTTP 版本，比如 HTTP/1.1，要求有效的请求或响应报文中必须包含特定的首部。本章稍后会探讨各种 HTTP 首部。

- 实体的主体部分（entity-body）
  实体的主体部分包含一个由任意数据组成的数据块。并不是所有的报文都包含实体的主体部分，有时，报文只是以一个 CRLF 结束。第 15 章详述了实体。

图 3-5 展示了一些假想的请求和响应报文。

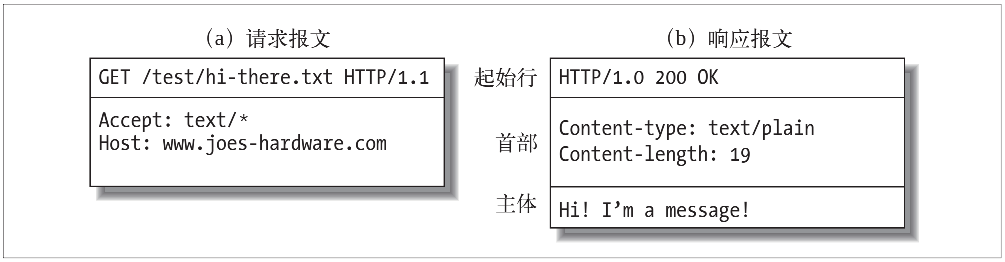


图 3-5　请求和响应报文示例

注意，一组 HTTP 首部总是应该以一个空行（仅有 CRLF）结束，甚至即使没有首部和实体的主体部分也应如此。但由于历史原因，很多客户端和服务器都在没有实体的主体部分时，（错误地）省略了最后的 CRLF。为了与这些流行但不符合规则的实现进行互通，客户端和服务器都应该接受那些没有最后那个 CRLF 的报文。

3.2.2　起始行

所有的 HTTP 报文都以一个起始行作为开始。请求报文的起始行说明了要做些什么。
47 响应报文的起始行说明发生了什么。

1. 请求行

请求报文请求服务器对资源进行一些操作。请求报文的起始行，或称为请求行，包含了一个方法和一个请求 URL，这个方法描述了服务器应该执行的操作，请求 URL 描述了要对哪个资源执行这个方法。请求行中还包含 HTTP 的版本，用来告知服务器，客户端使用的是哪种 HTTP。

所有这些字段都由空格符分隔。在图 3-5a 中，请求方法为 GET，请求 URL 为

/test/hi-there.txt，版本为 HTTP/1.1。在 HTTP/1.0 之前，并不要求请求行中包含 HTTP 版本号。

2. 响应行

响应报文承载了状态信息和操作产生的所有结果数据，将其返回给客户端。响应报文的起始行，或称为*响应行*，包含了响应报文使用的 HTTP 版本、数字状态码，以及描述操作状态的文本形式的原因短语。

所有这些字段都由空格符进行分隔。在图 3-5b 中，HTTP 版本为 HTTP/1.0，状态码为 200（表示成功），原因短语为 OK，表示文档已经被成功返回了。在 HTTP/1.0 之前，并不要求在响应中包含响应行。

3. 方法

请求的起始行以方法作为开始，方法用来告知服务器要做些什么。比如，在行“GET /specials/saw-blade.gif HTTP/1.0”中，方法就是 GET。

HTTP 规范中定义了一组常用的请求方法。比如，GET 方法负责从服务器获取一个文档，POST 方法会向服务器发送需要处理的数据，OPTIONS 方法用于确定 Web 服务器的一般功能，或者 Web 服务器处理特定资源的能力。

表 3-1 描述了 7 种这样的方法。注意，有些方法的请求报文中有主体，有些则是无主体的请求。

表3-1　常用的HTTP方法

方　法	描　述	是否包含主体
GET	从服务器获取一份文档	否
HEAD	只从服务器获取文档的首部	否
POST	向服务器发送需要处理的数据	是
PUT	将请求的主体部分存储在服务器上	是
TRACE	对可能经过代理服务器传送到服务器上去的报文进行追踪	否
OPTIONS	决定可以在服务器上执行哪些方法	否
DELETE	从服务器上删除一份文档	否

48

并不是所有服务器都实现了表 3-1 列出的所有 7 种方法。而且，由于 HTTP 设计得易于扩展，所以除了这些方法之外，其他服务器可能还会实现一些自己的请求方法。这些附加的方法是对 HTTP 规范的扩展，因此被称为*扩展方法*。

4. 状态码

方法是用来告诉服务器做什么事情的，状态码则用来告诉客户端，发生了什么事情。

状态码位于响应的起始行中。比如，在行 HTTP/1.0 200 OK 中，状态码就是 200。

客户端向一个 HTTP 服务器发送请求报文时，会发生很多事情。幸运的话，请求会成功完成。但你不会总是那么幸运的。服务器可能会告诉你无法找到所请求的资源，你没有访问资源的权限，或者资源被移到了其他地方。

状态码是在每条响应报文的起始行中返回的。会返回一个数字状态和一个可读的状态。数字码便于程序进行差错处理，而原因短语则更便于人们理解。

可以通过三位数字代码对不同状态码进行分类。200 到 299 之间的状态码表示成功。300 到 399 之间的代码表示资源已经被移走了。400 到 499 之间的代码表示客户端的请求出错了。500 到 599 之间的代码表示服务器出错了。

表 3-2 列出了状态码的分类。

表3-2　状态码分类

整体范围	已定义范围	分　　类
100 ～ 199	100 ～ 101	信息提示
200 ～ 299	200 ～ 206	成功
300 ～ 399	300 ～ 305	重定向
400 ～ 499	400 ～ 415	客户端错误
500 ～ 599	500 ～ 505	服务器错误

当前的 HTTP 版本只为每类状态定义了几个代码。随着协议的发展，HTTP 规范中会正式地定义更多的状态码。如果收到了不认识的状态码，可能是有人将其作为当前协议的扩展定义的。可以根据其所处范围，将它作为那个类别中一个普通的成员来处理。

比如，如果收到了状态码 515（在表 3-2 所列 5*XX* 代码的已定义范围之外），就应该
49 认为这条响应指出了服务器的错误，这是 5*XX* 报文的通用类别。

表 3-3 列出了部分最常见的状态码。本章稍后会详细解释当前在用的所有 HTTP 状态码。

表3-3　常见状态码

状　态　码	原因短语	含　　义
200	OK	成功。请求的所有数据都在响应主体中
401	Unauthorized（未授权）	需要输入用户名和密码
404	Not Found（未找到）	服务器无法找到所请求 URL 对应的资源

5. 原因短语

原因短语是响应起始行中的最后一个组件。它为状态码提供了文本形式的解释。比如，在行 HTTP/1.0 200 OK 中，OK 就是原因短语。

原因短语和状态码是成对出现的。原因短语是状态码的可读版本，应用程序开发者将其传送给用户，用以说明在请求期间发生了什么情况。

HTTP 规范并没有提供任何硬性规定，要求原因短语以何种形式出现。本章稍后列出了状态码和一些建议使用的原因短语。

6. 版本号

版本号会以 HTTP/*x.y* 的形式出现在请求和响应报文的起始行中。为 HTTP 应用程序提供了一种将自己所遵循的协议版本告知对方的方式。

使用版本号的目的是为使用 HTTP 的应用程序提供一种线索，以便互相了解对方的能力和报文格式。在与使用 HTTP 1.1 的应用程序进行通信的 HTTP 1.2 应用程序应该知道，它不能使用任何新的 1.2 特性，因为使用老版本协议的应用程序很可能无法实现这些特性。

版本号说明了应用程序支持的最高 HTTP 版本。但 HTTP/1.0 应用程序在解释包含 HTTP/1.1 的响应时，会认为这个响应是个 1.1 响应，而实际上这只是响应应用程序所使用的协议等级，在这些情况下，版本号会在应用程序之间造成误解 [2]。

注意，版本号不会被当作小数来处理。版本中的每个数字（比如 HTTP/1.0 中的 1 和 0）都会被当作一个单独的数字来处理。因此，在比较 HTTP 版本时，每个数字 50
都必须单独进行比较，以便确定哪个版本更高。比如，HTTP/2.22 就比 HTTP/2.3 的版本要高，因为 22 比 3 大。

3.2.3 首部

前一小节的重点是请求和响应报文的第一行（方法、状态码、原因短语和版本号）。跟在起始行后面的就是零个、一个或多个 HTTP 首部字段（参见图 3-5）。

HTTP 首部字段向请求和响应报文中添加了一些附加信息。本质上来说，它们只是一些名 / 值对的列表。比如，下面的首部行会向 `Content-Length` 首部字段赋值 19：

```
Content-length：19
```

注 2：http://httpd.apache.org/docs-2.0/misc/known_client_problems.html 上有更多在 Apache 与客户端之间出现此问题的案例。

1. 首部分类

HTTP 规范定义了几种首部字段。应用程序也可以随意发明自己所用的首部。HTTP 首部可以分为以下几类。

- 通用首部

  既可以出现在请求报文中，也可以出现在响应报文中。

- 请求首部

  提供更多有关请求的信息。

- 响应首部

  提供更多有关响应的信息。

- 实体首部

  描述主体的长度和内容，或者资源自身。

- 扩展首部

  规范中没有定义的新首部。

每个 HTTP 首部都有一种简单的语法：名字后面跟着冒号（：），然后跟上可选的空格，再跟上字段值，最后是一个 CRLF。表 3-4 列出了一些常见的首部实例。

表3-4　常见的首部实例

首部实例	描　　述
`Date:Tue,3Oct 1997 02:16:03 GMT`	服务器产生响应的日期
`Content-length:15040`	实体的主体部分包含了 15 040 字节的数据
`Content-type:image/gif`	实体的主体部分是一个 GIF 图片
`Accept: image/gif, image/jpeg, text/html`	客户端可以接收 GIF 图片和 JPEG 图片以及 HTML

2. 首部延续行

将长的首部行分为多行可以提高可读性，多出来的每行前面至少要有一个空格或制
51 表符（tab）。

例如：

```
HTTP/1.0 200 OK
Content-Type: image/gif
Content-Length: 8572
```

```
Server: Test Server
    Version 1.0
```

在这个例子中，响应报文里包含了一个 `Server` 首部，其值被划分成了多个延续行。该首部的完整值为 `Test Server Version 1.0`。

本章稍后将简要介绍所有的 HTTP 首部。附录 C 提供了所有首部更为详细的参考。

3.2.4 实体的主体部分

HTTP 报文的第三部分是可选的实体主体部分。实体的主体是 HTTP 报文的负荷。就是 HTTP 要传输的内容。

HTTP 报文可以承载很多类型的数字数据：图片、视频、HTML 文档、软件应用程序、信用卡事务、电子邮件等。

3.2.5 版本0.9的报文

HTTP 版本 0.9 是 HTTP 协议的早期版本。是当今 HTTP 所拥有的请求及响应报文的鼻祖，但其协议要简单得多（参见图 3-6）。

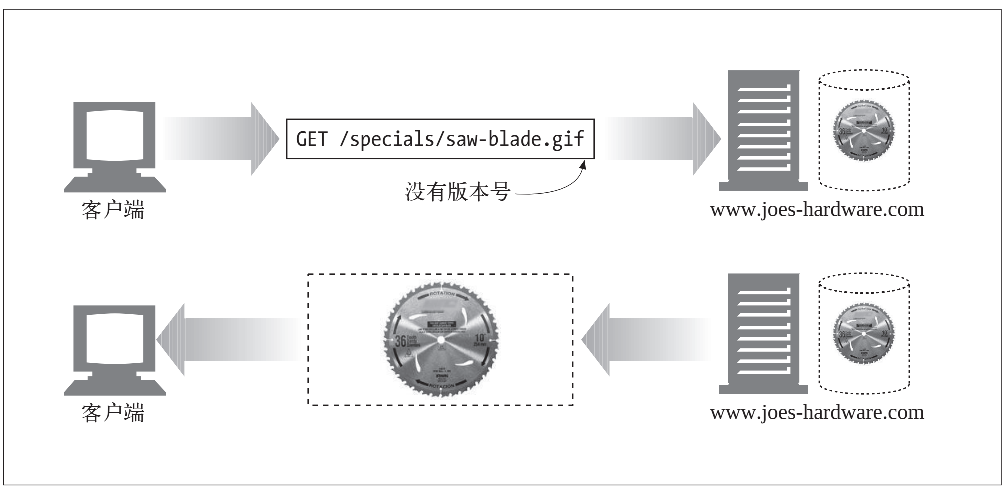


图 3-6　HTTP/0.9 事务

HTTP/0.9 报文也由请求和响应组成，但请求中只包含方法和请求 URL，响应中只包含实体。它没有版本信息（它是第一个，而且是当时唯一的版本），没有状态码或原因短语，也没有首部。 52

但这种简单协议无法提供更多的灵活性，也无法实现本书中描述的大部分 HTTP 特性和应用。这里对其进行简要的描述，是因为仍然有一些客户端、服务器和其他应用程序在使用这个协议，应用程序的编写者应该清楚它的局限性。

3.3 方法

现在，我们对前面在表 3-1 中列出的一些基本 HTTP 方法进行更为深入的讨论。注意，并不是每个服务器都实现了所有的方法。如果一台服务器要与 HTTP 1.1 兼容，那么只要为其资源实现 GET 方法和 HEAD 方法就可以了。

即使服务器实现了所有这些方法，这些方法的使用很可能也是受限的。例如，支持 DELETE 方法或 PUT 方法（本节稍后介绍）的服务器可能并不希望任何人都能够删除或存储资源。这些限制通常都是在服务器的配置中进行设置的，因此会随着站点和服务器的不同而有所不同。

3.3.1 安全方法

HTTP 定义了一组被称为安全方法的方法。GET 方法和 HEAD 方法都被认为是安全的，这就意味着使用 GET 或 HEAD 方法的 HTTP 请求都不会产生什么动作。

不产生动作，在这里意味着 HTTP 请求不会在服务器上产生什么结果。例如，你在 Joe 的五金商店购物时，点击了“提交购买”按钮。点击按钮时会提交一个带有信用卡信息的 POST 请求（稍后讨论），那么在服务器上，就会为你执行一个动作。在这种情况下，为购买行为支付信用卡就是所执行的动作。

安全方法并不一定是什么动作都不执行的（实际上，这是由 Web 开发者决定的）。使用安全方法的目的就是当使用可能引发某一动作的不安全方法时，允许 HTTP 应用程序开发者通知用户。在 Joe 的五金商店的例子中，你的 Web 浏览器可能会弹出一条警告消息，说明你正在用不安全的方法发起请求，这样可能会在服务器上引发一些事件（比如用你的信用卡支付费用）。

3.3.2 GET

GET 是最常用的方法。通常用于请求服务器发送某个资源。HTTP/1.1 要求服务器实现此方法。图 3-7 显示了一个例子，在这个例子中，客户端用 GET 方法发起了一
53 次 HTTP 请求。

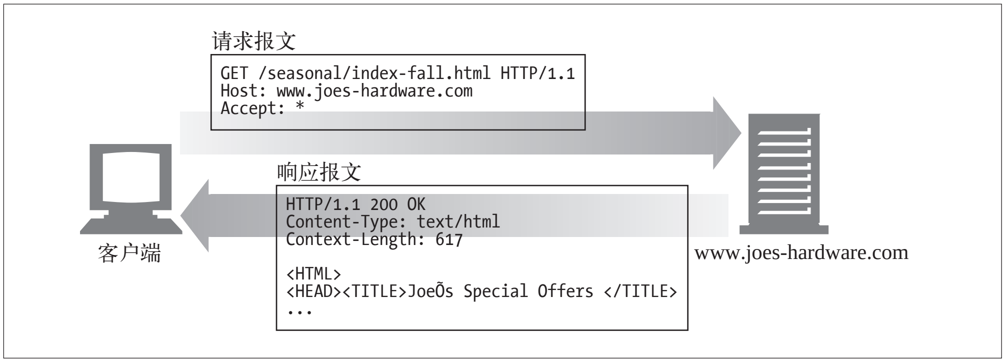


图 3-7　GET 示例

3.3.3　HEAD

HEAD 方法与 GET 方法的行为很类似，但服务器在响应中只返回首部。不会返回实体的主体部分。这就允许客户端在未获取实际资源的情况下，对资源的首部进行检查。使用 HEAD，可以：

- 在不获取资源的情况下了解资源的情况（比如，判断其类型）；
- 通过查看响应中的状态码，看看某个对象是否存在；
- 通过查看首部，测试资源是否被修改了。

服务器开发者必须确保返回的首部与 GET 请求所返回的首部完全相同。遵循 HTTP/1.1 规范，就必须实现 HEAD 方法。图 3-8 显示了实际的 HEAD 方法。

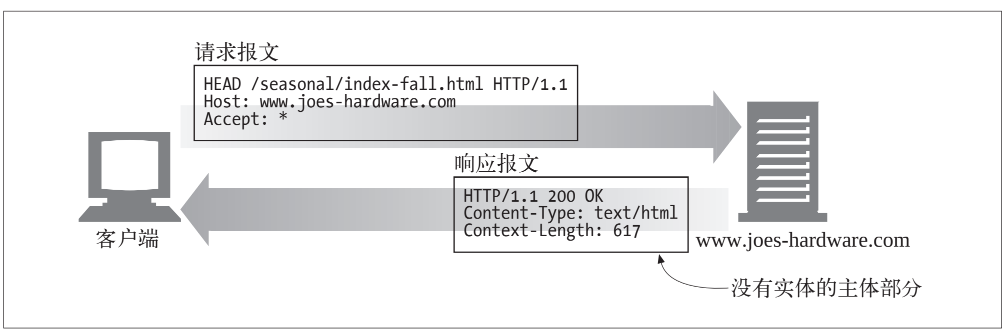


图 3-8　HEAD 示例

3.3.4　PUT

与 GET 从服务器读取文档相反，PUT 方法会向服务器写入文档。有些发布系统允许用户创建 Web 页面，并用 PUT 直接将其安装到 Web 服务器上去（参见图 3-9）。 54

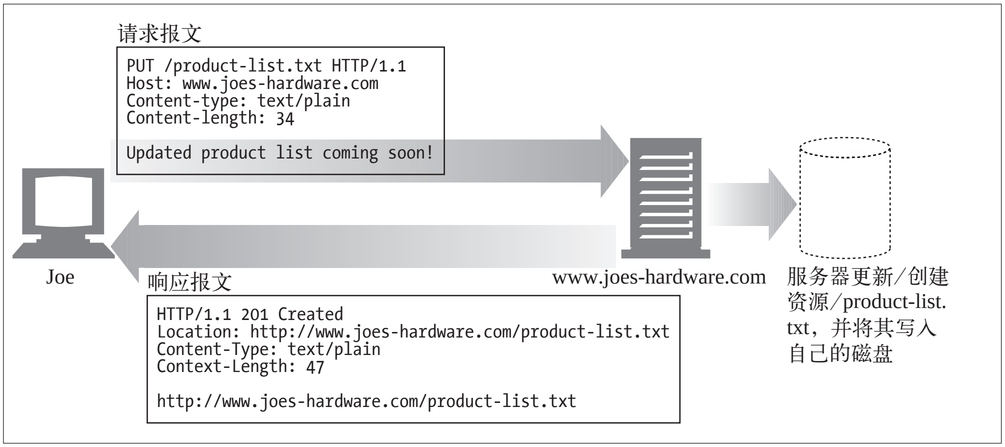


图 3-9　PUT 示例

PUT 方法的语义就是让服务器用请求的主体部分来创建一个由所请求的 URL 命名的新文档，或者，如果那个 URL 已经存在的话，就用这个主体来替代它。

因为 PUT 允许用户对内容进行修改，所以很多 Web 服务器都要求在执行 PUT 之前，用密码登录。在第 12 章中可以读到更多有关密码认证的内容。

3.3.5　POST

POST 方法起初是用来向服务器输入数据的[3]。实际上，通常会用它来支持 HTML 的表单。表单中填好的数据通常会被送给服务器，然后由服务器将其发送到它要去的地方（比如，送到一个服务器网关程序中，然后由这个程序对其进行处理）。图 3-10 显示了一个用 POST 方法发起 HTTP 请求——向服务器发送表单数据——的客户端。

3.3.6　TRACE

客户端发起一个请求时，这个请求可能要穿过防火墙、代理、网关或其他一些应用程序。每个中间节点都可能会修改原始的 HTTP 请求。TRACE 方法允许客户端在最终将请求发送给服务器时，看看它变成了什么样子。

TRACE 请求会在目的服务器端发起一个“环回”诊断。行程最后一站的服务器会弹回一条 TRACE 响应，并在响应主体中携带它收到的原始请求报文。这样客户端
55 就可以查看在所有中间 HTTP 应用程序组成的请求 / 响应链上，原始报文是否，以及如何被毁坏或修改过（参见图 3-11）。

注 3：POST 用于向服务器发送数据。PUT 用于向服务器上的资源（例如文件）中存储数据。

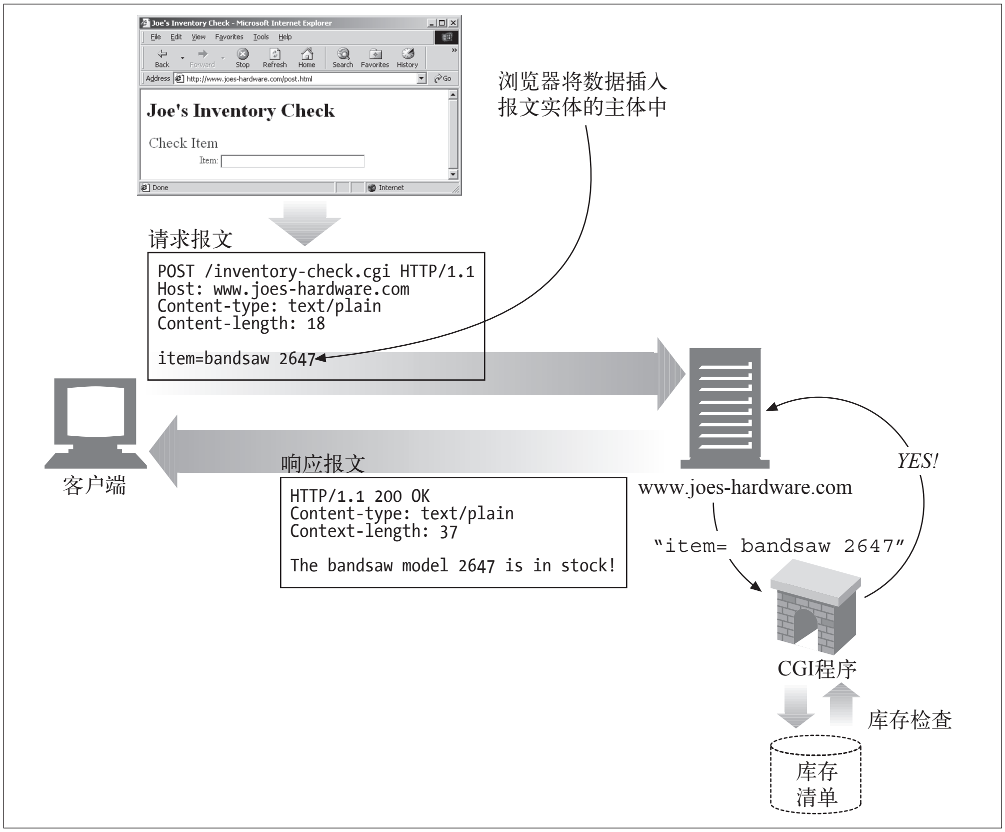


图 3-10　POST 示例

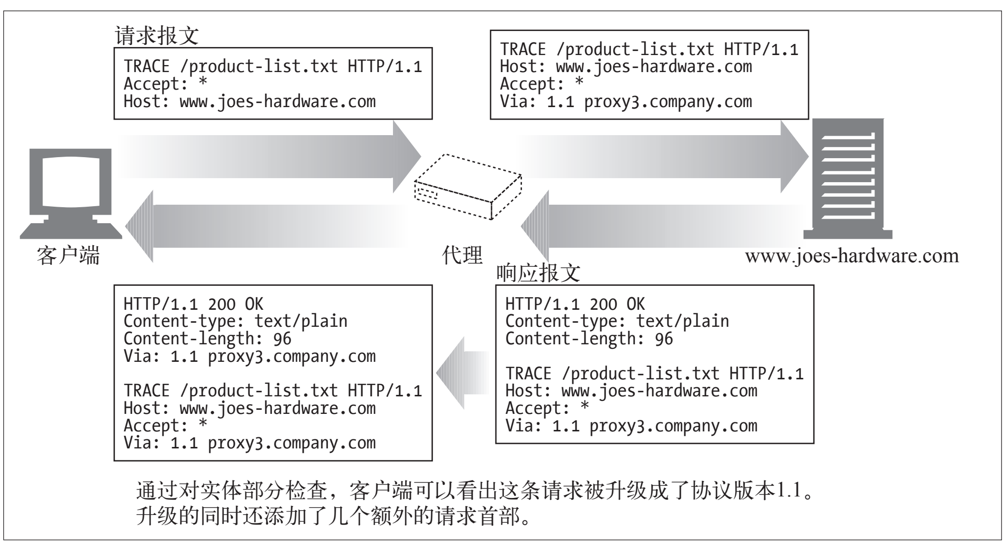


图 3-11　TRACE 示例

TRACE 方法主要用于诊断；也就是说，用于验证请求是否如愿穿过了请求 / 响应链。它也是一种很好的工具，可以用来查看代理和其他应用程序对用户请求所产生效果。

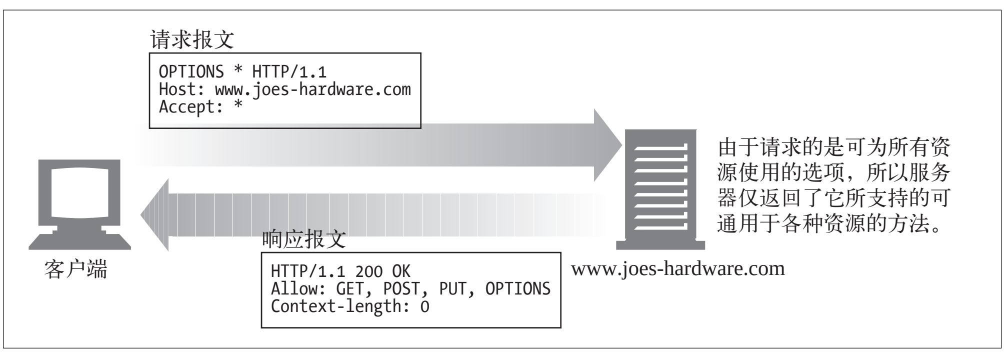
尽管 TRACE 可以很方便地用于诊断，但它确实也有缺点，它假定中间应用程序对各种不同类型请求（不同的方法——GET、HEAD、POST 等）的处理是相同的。很多 HTTP 应用程序会根据方法的不同做出不同的事情——比如，代理可能会将 POST 请求直接发送给服务器，而将 GET 请求发送给另一个 HTTP 应用程序（比如 Web 缓存）。TRACE 并不提供区分这些方法的机制。通常，中间应用程序会自行决
56 定对 TRACE 请求的处理方式。

TRACE 请求中不能带有实体的主体部分。TRACE 响应的实体主体部分包含了响应服务器收到的请求的精确副本。

3.3.7 OPTIONS

OPTIONS 方法请求 Web 服务器告知其支持的各种功能。可以询问服务器通常支持哪些方法，或者对某些特殊资源支持哪些方法。(有些服务器可能只支持对一些特殊类型的对象使用特定的操作)。

这为客户端应用程序提供了一种手段，使其不用实际访问那些资源就能判定访问各种资源的最优方式。图 3-12 显示了一个使用 OPTIONS 方法的请求。



57 图 3-12 OPTIONS 示例

3.3.8 DELETE

顾名思义，DELETE 方法所做的事情就是请服务器删除请求 URL 所指定的资源。但是，客户端应用程序无法保证删除操作一定会被执行。因为 HTTP 规范允许服务器在不通知客户端的情况下撤销请求。图 3-13 显示了一个 DELETE 方法实例。

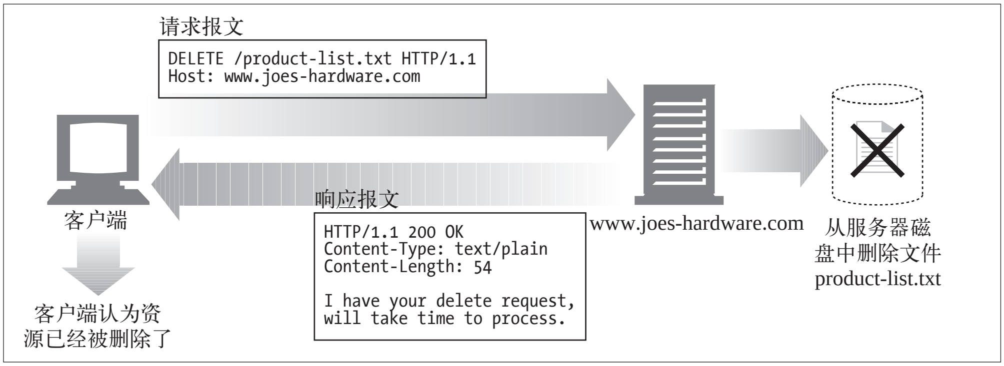


图 3-13　DELETE 示例

3.3.9　扩展方法

HTTP 被设计成字段可扩展的，这样新的特性就不会使老的软件失效了。扩展方法指的就是没有在 HTTP/1.1 规范中定义的方法。服务器会为它所管理的资源实现一些 HTTP 服务，这些方法为开发者提供了一种扩展这些 HTTP 服务能力的手段。表 3-5 列出了一些常见的扩展方法实例。这些方法就是 WebDAV HTTP 扩展（参见第 19 章）包含的所有方法，这些方法有助于通过 HTTP 将 Web 内容发布到 Web 服务器上去。

表3-5　Web发布扩展方法示例

方　法	描　述
LOCK	允许用户“锁定”资源——比如，可以在编辑某个资源的时候将其锁定，以防别人同时对其进行修改
MKCOL	允许用户创建资源
COPY	便于在服务器上复制资源
MOVE	在服务器上移动资源

并不是所有的扩展方法都是在正式规范中定义的，认识到这一点很重要。如果你定义了一个扩展方法，很可能大部分 HTTP 应用程序都无法理解。同样，你的 HTTP 应用程序也可能会遇到一些其他应用程序在用的，而它并不理解的扩展方法。 58

在这些情况下，最好对扩展方法宽容一些。如果能够在不破坏端到端行为的情况下将带有未知方法的报文传递给下游服务器，代理应尝试传递这些报文。如果可能破坏端到端行为则应以 501 Not Implemented（无法实现）状态码进行响应。最好按惯例“对所发送的内容要求严一点，对所接收的内容宽容一些”来处理扩展方法（以及一般的 HTTP 扩展）。

3.4 状态码

如前面的表 3-2 所示，HTTP 状态码被分成了五大类。本节对这五类 HTTP 状态码中的每一类都进行了总结。

状态码为客户端提供了一种理解事务处理结果的便捷方式。尽管并没有实际的规范对原因短语的确切文本进行说明，本节还是列出了一些原因短语示例。我们所列的是 HTTP/1.1 规范推荐使用的原因短语。

3.4.1 100 ~ 199——信息性状态码

HTTP/1.1 向协议中引入了信息性状态码。这些状态码相对较新，关于其复杂性和感知价值存在一些争论，而受到限制。表 3-6 列出了已定义的信息性状态码。

表3-6 信息性状态码及原因短语

状态码	原因短语	含义
100	Continue	说明收到了请求的初始部分，请客户端继续。发送了这个状态码之后，服务器在收到请求之后必须进行响应。更多信息请参见附录 C 中的 `Expect` 首部介绍
101	Switching Protocols	说明服务器正在根据客户端的指定，将协议切换成 `Update` 首部所列的协议

100 Continue 状态码尤其让人糊涂。它的目的是对这样的情况进行优化：HTTP 客户端应用程序有一个实体的主体部分要发送给服务器，但希望在发送之前查看一下服务器是否会接受这个实体。这可能会给 HTTP 程序员带来一些困扰，因此在这里进行了比较详细（它如何与客户端、服务器和代理进行通信）的讨论。

1. 客户端与100 Continue

如果客户端在向服务器发送一个实体，并且愿意在发送实体之前等待 100 Continue 响应，那么，客户端就要发送一个携带了值为 100 Continue 的 `Expect` 请求首部（参见附录 C）。如果客户端没有发送实体，就不应该发送 100 Continue `Expect` 首
59 部，因为这样会使服务器误以为客户端要发送一个实体。

从很多方面来看，100 Continue 都是一种优化。客户端应用程序只有在避免向服务器发送一个服务器无法处理或使用的大实体时，才应该使用 100 Continue。

由于起初对 100 Continue 状态存在一些困惑（而且以前有些实现在这里出过问题），因此发送了值为 100 Continue 的 `Expect` 首部的客户端不应该永远在那儿等待服务器发送 100 Continue 响应。超时一定时间之后，客户端应该直接将实体发送出去。

实际上，客户端程序的实现者也应该做好应对非预期 100 Continue 响应的准备（这很烦人，但确实如此）。有些出错的 HTTP 应用程序会不合时宜地发送这个状态码。

2. 服务器与100 Continue

如果服务器收到了一条带有值为 100 Continue 的 `Expect` 首部的请求，它会用 100 Continue 响应或一条错误码来进行响应（参见表 3-9）。服务器永远也不应该向没有发送 100 Continue 期望的客户端发送 100 Continue 状态码。但如前所述，有些出错的服务器可能会这么做。

如果出于某种原因，服务器在有机会发送 100 Continue 响应之前就收到了部分（或全部）的实体，就说明客户端已经决定继续发送数据了，这样，服务器就不需要发送这个状态码了。但服务器读完请求之后，还是应该为请求发送一个最终状态码（它可以跳过 100 Continue 状态）。

最后，如果服务器收到了带有 100 Continue 期望的请求，而且它决定在读取实体的主体部分之前（比如，因为出错而）结束请求，就不应该仅仅是发送一条响应并关闭连接，因为这样会妨碍客户端接收响应（参见 4.7.4 节）。

3. 代理与100 Continue

如果代理从客户端收到了一条带有 100 Continue 期望的请求，它需要做几件事情。如果代理知道下一跳服务器（在第 6 章中讨论）是 HTTP/1.1 兼容的，或者并不知道下一跳服务器与哪个版本兼容，它都应该将 `Expect` 首部放在请求中向下转发。如果它知道下一跳服务器只能与 HTTP/1.1 之前的版本兼容，就应该以 417 Expectation Failed 错误进行响应。[4]

如果代理决定代表与 HTTP/1.0 或之前版本兼容的客户端，在其请求中放入 `Expect` 首部和 100 Continue 值，那么，（如果它从服务器收到了 100 Continue 响应）它就不应该将 100 Continue 响应转发给客户端，因为客户端可能不知道该拿它怎么办。

代理维护一些有关下一跳服务器及其所支持的 HTTP 版本的状态信息（至少要维护那些最近收到过请求的服务器的相关状态）是有好处的，这样它们就可以更好地处理收到的那些带有 100 Continue 期望的请求了。 60

3.4.2　200～299——成功状态码

客户端发起请求时，这些请求通常都是成功的。服务器有一组用来表示成功的状态码，分别对应于不同类型的请求。表 3-7 列出了已定义的成功状态码。

注 4：还有一种合理的方法，是向客户端先返回 100 Continue，在向服务器转发请求时，删掉 `Expect` 首部。（译者注）

表3-7　成功状态码和原因短语

状态码	原因短语	含　义
200	OK	请求没问题，实体的主体部分包含了所请求的资源
201	Created	用于创建服务器对象的请求（比如，PUT）。响应的实体主体部分中应该包含各种引用了已创建的资源的 URL，Location 首部包含的则是最具体的引用。更多有关 Location 首部的信息参见表 3-21。 服务器必须在发送这个状态码之前创建好对象
202	Accepted	请求已被接受，但服务器还未对其执行任何动作。不能保证服务器会完成这个请求；这只是意味着接受请求时，它看起来是有效的。 服务器应该在实体的主体部分包含对请求状态的描述，或许还应该有对请求完成时间的估计（或者包含一个指针，指向可以获取此信息的位置）
203	Non-Authoritative Information	实体首部（更多有关实体首部的信息参见 3.5.4 节）包含的信息不是来自于源端服务器，而是来自资源的一份副本。如果中间节点上有一份资源副本，但无法或者没有对它所发送的与资源有关的元信息（首部）进行验证，就会出现这种情况。 这种响应码并不是非用不可的；如果实体首部来自源端服务器，响应为 200 状态的应用程序就可以将其作为一种可选项使用
204	No Content	响应报文中包含若干首部和一个状态行，但没有实体的主体部分。主要用于在浏览器不转为显示新文档的情况下，对其进行更新（比如刷新一个表单页面）
205	Reset Content	另一个主要用于浏览器的代码。负责告知浏览器清除当前页面中的所有 HTML 表单元素
206	Partial Content	成功执行了一个部分或 Range（范围）请求。稍后我们会看到，客户端可以通过一些特殊的首部来获取部分或某个范围内的文档——这个状态码就说明范围请求成功了。更多有关 Range 首部的内容参见 15.9 节。 206 响应中必须包含 Content-Range、Date 以及 ETag 或 Content-Location 首部

3.4.3　300 ~ 399——重定向状态码

重定向状态码要么告知客户端使用替代位置来访问他们所感兴趣的资源，要么就提供一个替代的响应而不是资源的内容。如果资源已被移动，可发送一个重定向状态
61 码和一个可选的 Location 首部来告知客户端资源已被移走，以及现在可以在哪里找到它（参见图 3-14）。这样，浏览器就可以在不打扰使用者的情况下，透明地转入新的位置了。

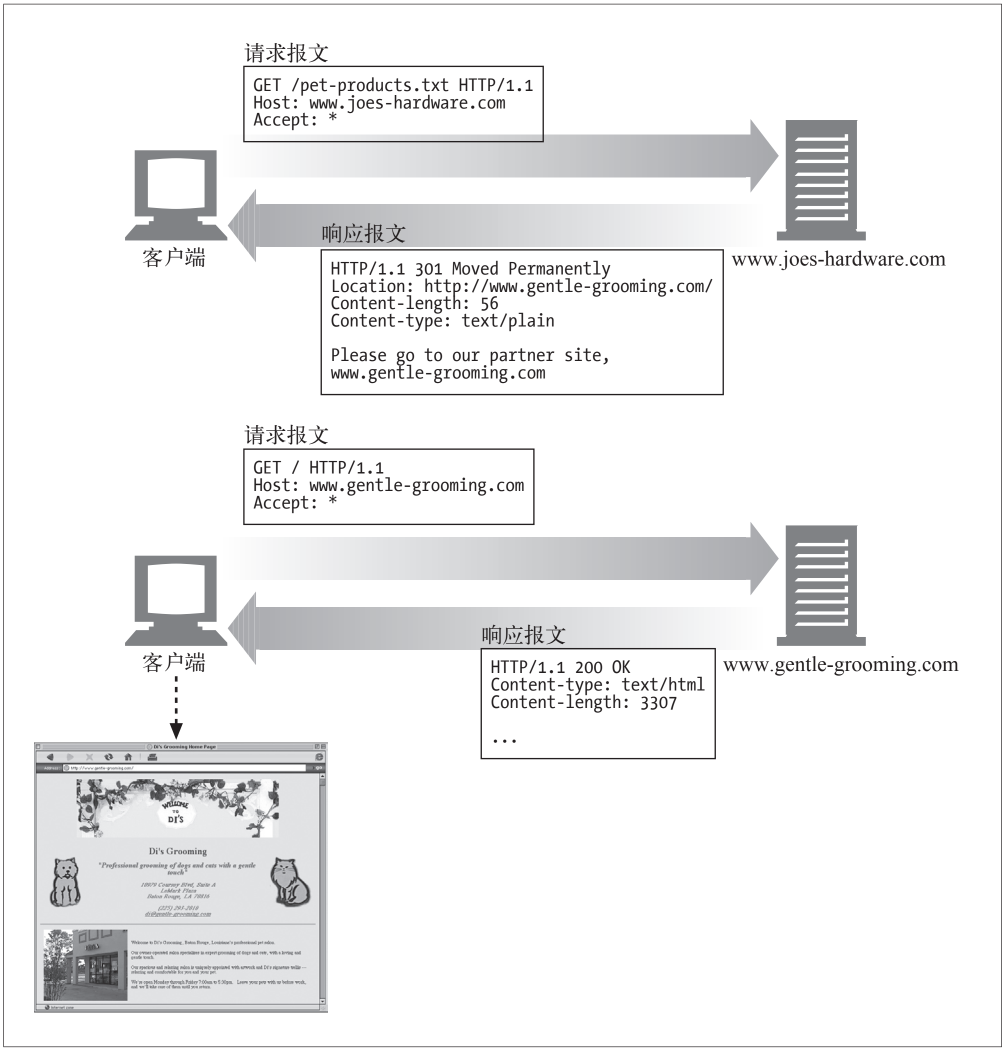


图 3-14　将请求重定向到新的位置

可以通过某些重定向状态码对资源的应用程序本地副本与源端服务器上的资源进行验证。比如，HTTP 应用程序可以查看其资源的本地副本是否仍然是最新的，或者在源端服务器上资源是否被修改过。图 3-15 显示了一个这样的例子。客户端发送了一个特殊的 `If-Modified-Since` 首部，说明只读取 1997 年 10 月之后修改过的文档。这个日期之后，此文档并未被修改过，因此，服务器回送了一个 304 状态码，而不是文档的内容。 62

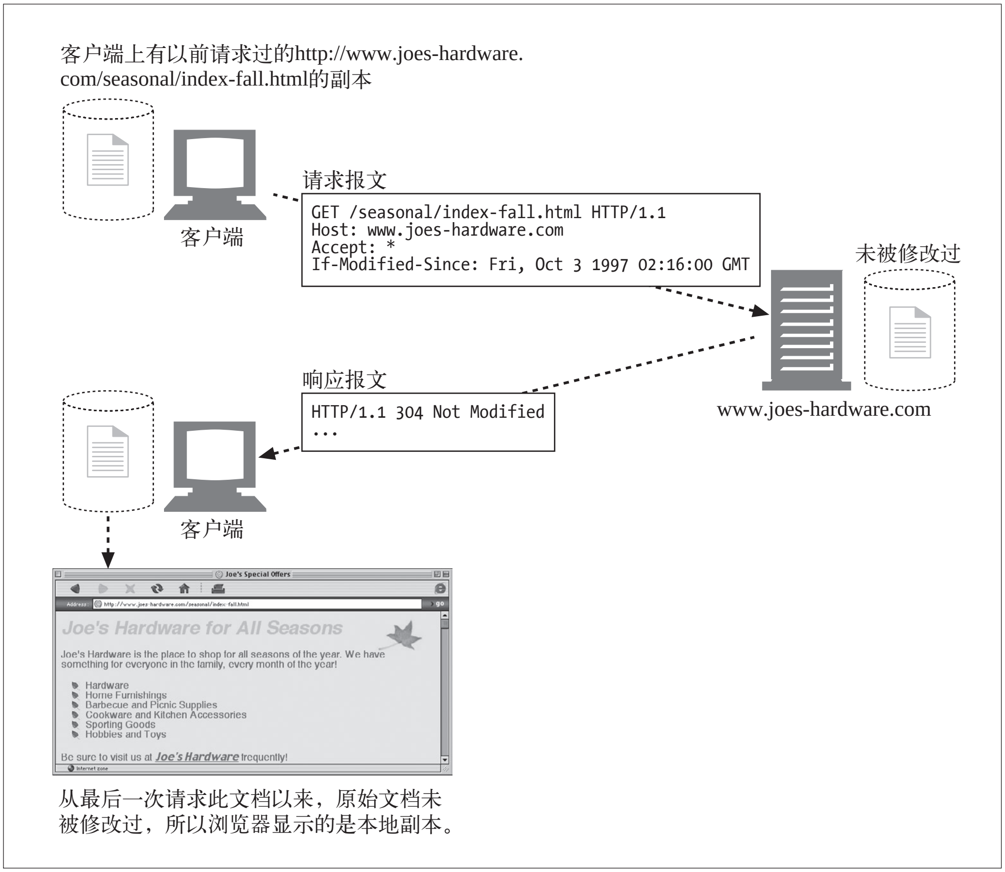


图 3-15　重定向为使用本地副本的请求

总之，在对那些包含了重定向状态码的非 HEAD 请求进行响应时，最好要包含一个实体，并在实体中包含描述信息和指向（多个）重定向 URL 的链接——参见图 3-14 的第一个响应报文。表 3-8 列出了已定义的重定向状态码。

表3-8　重定向状态码与原因短语

状态码	原因短语	含　　义
300	Multiple Choices	客户端请求一个实际指向多个资源的 URL 时会返回这个状态码，比如服务器上有某个 HTML 文档的英语和法语版本。返回这个代码时会带有一个选项列表；这样用户就可以选择他希望使用的那一项了。有多个版本可用时，客户端需要沟通解决，更多与此有关的信息请参见第 17 章。服务器可以在 `Location` 首部包含首选 URL
301	Moved Permanently	在请求的 URL 已被移除时使用。响应的 `Location` 首部中应该包含资源现在所处的 URL

（续）

状态码	原因短语	含　　义
302	Found	与 301 状态码类似；但是，客户端应该使用 Location 首部给出的 URL 来临时定位资源。将来的请求仍应使用老的 URL
303	See Other	告知客户端应该用另一个 URL 来获取资源。新的 URL 位于响应报文的 Location 首部。其主要目的是允许 POST 请求的响应将客户端定向到某个资源上去
304	Not Modified	客户端可以通过所包含的请求首部，使其请求变成有条件的。更多有关条件首部的内容请参见第 3 章。如果客户端发起了一个条件 GET 请求，而最近资源未被修改的话，就可以用这个状态码来说明资源未被修改。带有这个状态码的响应不应该包含实体的主体部分
305	Use Proxy	用来说明必须通过一个代理来访问资源；代理的位置由 Location 首部给出。很重要的一点是，客户端是相对某个特定资源来解析这条响应的，不能假定所有请求，甚至所有对持有所请求资源的服务器的请求都通过这个代理进行。如果客户端错误地让代理介入了某条请求，可能会引发破坏性的行为，而且会造成安全漏洞
306	（未使用）	当前未使用
307	Temporary Redirect	与 301 状态码类似；但客户端应该使用 Location 首部给出的 URL 来临时定位资源。将来的请求应该使用老的 URL

63

从表 3-8 中，你可能已经注意到 302、303 和 307 状态码之间存在一些交叉。这些状态码的用法有着细微的差别，大部分差别都源于 HTTP/1.0 和 HTTP/1.1 应用程序对这些状态码处理方式的不同。

当 HTTP/1.0 客户端发起一个 POST 请求，并在响应中收到 302 重定向状态码时，它会接受 Location 首部的重定向 URL，并向那个 URL 发起一个 GET 请求（而不会像原始请求中那样发起 POST 请求）。

HTTP/1.0 服务器希望 HTTP/1.0 客户端这么做——如果 HTTP/1.0 服务器收到来自 HTTP/1.0 客户端的 POST 请求之后发送了 302 状态码，服务器就期望客户端能够接受重定向 URL，并向重定向的 URL 发送一个 GET 请求。

问题出在 HTTP/1.1。HTTP/1.1 规范使用 303 状态码来实现同样的行为（服务器发送 303 状态码来重定向客户端的 POST 请求，在它后面跟上一个 GET 请求）。

为了避开这个问题，HTTP/1.1 规范指出，对于 HTTP/1.1 客户端，用 307 状态码取代 302 状态码来进行临时重定向。这样服务器就可以将 302 状态码保留起来，为 HTTP/1.0 客户端使用了。

这样一来，服务器要选择适当的重定向状态码放入重定向响应中发送，就需要查看
64 客户端的 HTTP 版本了。

3.4.4　400～499——客户端错误状态码

有时客户端会发送一些服务器无法处理的东西，比如格式错误的请求报文，或者最常见的是，请求一个不存在的 URL。

浏览网页时，我们都看到过臭名昭著的 404 Not Found 错误码——这只是服务器在告诉我们，它对我们请求的资源一无所知。

很多客户端错误都是由浏览器来处理的，甚至不会打扰到你。只有少量错误，比如 404，还是会穿过浏览器来到用户面前。表 3-9 显示了各种客户端的错误状态码。

表3-9　客户端错误状态码及原因短语

状态码	原因短语	含　义
400	Bad Request	用于告知客户端它发送了一个错误的请求
401	Unauthorized	与适当的首部一同返回，在这些首部中请求客户端在获取对资源的访问权之前，对自己进行认证。更多有关认证的内容请参见 12.1 节
402	Payment Required	现在这个状态码还未使用，但已经被保留，以作未来之用
403	Forbidden	用于说明请求被服务器拒绝了。如果服务器想说明为什么拒绝请求，可以包含实体的主体部分来对原因进行描述。但这个状态码通常是在服务器不想说明拒绝原因的时候使用的
404	Not Found	用于说明服务器无法找到所请求的 URL。通常会包含一个实体，以便客户端应用程序显示给用户看
405	Method Not Allowed	发起的请求中带有所请求的 URL 不支持的方法时，使用此状态码。应该在响应中包含 `Allow` 首部，以告知客户端对所请求的资源可以使用哪些方法。更多有关 `Allow` 首部的信息请参见 3.5.4 节
406	Not Acceptable	客户端可以指定参数来说明它们愿意接收什么类型的实体。服务器没有与客户端可接受的 URL 相匹配的资源时，使用此代码。通常，服务器会包含一些首部，以便客户端弄清楚为什么请求无法满足。更多信息请参见第 17 章
407	Proxy Authentication Required	与 401 状态码类似，但用于要求对资源进行认证的代理服务器
408	Request Timeout	如果客户端完成请求所花的时间太长，服务器可以回送此状态码，并关闭连接。超时时长随服务器的不同有所不同，但通常对所有的合法请求来说，都是够长的
409	Conflict	用于说明请求可能在资源上引发的一些冲突。服务器担心请求会引发冲突时，可以发送此状态码。响应中应该包含描述冲突的主体

（续）

65

状态码	原因短语	含　　义
410	Gone	与 404 类似，只是服务器曾经拥有过此资源。主要用于 Web 站点的维护，这样服务器的管理者就可以在资源被移除的情况下通知客户端了
411	Length Required	服务器要求在请求报文中包含 `Content-Length` 首部时使用。更多有关 `Content-Length` 首部的信息请参见 3.5.4 节
412	Precondition Failed	客户端发起了条件请求，且其中一个条件失败了的时候使用。客户端包含了 `Expect` 首部时发起的就是条件请求。更多有关 `Expect` 首部的内容请参见附录 C 中 `Expect` 部分
413	Request Entity Too Large	客户端发送的实体主体部分比服务器能够或者希望处理的要大时，使用此状态码
414	Request URI Too Long	客户端所发请求中的请求 URL 比服务器能够或者希望处理的要长时，使用此状态码
415	Unsupported Media Type	服务器无法理解或无法支持客户端所发实体的内容类型时，使用此状态码
416	Requested Range Not Satisfiable	请求报文所请求的是指定资源的某个范围，而此范围无效或无法满足时，使用此状态码
417	Expectation Failed	请求的 `Expect` 请求首部包含了一个期望，但服务器无法满足此期望时，使用此状态码。更多有关 `Expect` 首部的内容请参见附录 C 中 `Expect` 部分 如果代理或其他中间应用程序有确切证据说明源端服务器会为某请求产生一个失败的期望，就可以发送这个响应状态码

3.4.5　500 ~ 599——服务器错误状态码

有时客户端发送了一条有效请求，服务器自身却出错了。这可能是客户端碰上了服务器的缺陷，或者服务器上的子元素，比如某个网关资源，出了错。

代理尝试着代表客户端与服务器进行交流时，经常会出现问题。代理会发布 5XX 服务器错误状态码来描述所遇到的问题（参见第 6 章）。表 3-10 列出了已定义的服务器错误状态码。

表3-10　服务器错误状态码及原因短语

状态码	原因短语	含　　义
500	Internal Server Error	服务器遇到一个妨碍它为请求提供服务的错误时，使用此状态码
501	Not Implemented	客户端发起的请求超出服务器的能力范围（比如，使用了服务器不支持的请求方法）时，使用此状态码

（续）

状态码	原因短语	含　　义
502	Bad Gateway	作为代理或网关使用的服务器从请求响应链的下一条链路上收到了一条伪响应（比如，它无法连接到其父网关）时，使用此状态码
503	Service Unavailable	用来说明服务器现在无法为请求提供服务，但将来可以。如果服务器知道什么时候资源会变为可用的，可以在响应中包含一个 `Retry-After` 首部。更多有关 `Retry-After` 首部的信息请参见 3.5.3 节
504	Gateway Timeout	与状态码 408 类似，只是这里的响应来自一个网关或代理，它们在等待另一服务器对其请求进行响应时超时了
505	HTTP Version Not Supported	服务器收到的请求使用了它无法或不愿支持的协议版本时，使用此状态码。有些服务器应用程序会选择不支持协议的早期版本

66

3.5　首部

首部和方法配合工作，共同决定了客户端和服务器能做什么事情。本节快速介绍了使用标准 HTTP 首部及一些没有在 HTTP/1.1 规范（RFC 2616）中明确定义的首部的目的。附录 C 对所有这些首部进行了更详细的总结。

在请求和响应报文中都可以用首部来提供信息，有些首部是某种报文专用的，有些首部则更通用一些。可以将首部分为五个主要的类型。

- *通用首部*
  这些是客户端和服务器都可以使用的通用首部。可以在客户端、服务器和其他应用程序之间提供一些非常有用的通用功能。比如，`Date` 首部就是一个通用首部，每一端都可以用它来说明构建报文的时间和日期：

  ```
  Date: Tue, 3 Oct 1974 02:16:00 GMT
  ```

- *请求首部*
  从名字中就可以看出，请求首部是请求报文特有的。它们为服务器提供了一些额外信息，比如客户端希望接收什么类型的数据。例如，下面的 `Accept` 首部就用来告知服务器客户端会接受与其请求相符的任意媒体类型：

  ```
  Accept: */*
  ```

- *响应首部*
  响应报文有自己的首部集，以便为客户端提供信息（比如，客户端在与哪种类型的服务器进行交互）。例如，下列 `Server` 首部就用来告知客户端它在与一个版

本 1.0 的 Tiki-Hut 服务器进行交互：

```
Server: Tiki-Hut/1.0
```

- 实体首部

  实体首部指的是用于应对实体主体部分的首部。比如，可以用实体首部来说明实体主体部分的数据类型。例如，可以通过下列 `Content-Type` 首部告知应用程序，数据是以 iso-latin-1 字符集表示的 HTML 文档：

  ```
  Content-Type: text/html; charset=iso-latin-1
  ```
  67

- 扩展首部

  扩展首部是非标准的首部，由应用程序开发者创建，但还未添加到已批准的 HTTP 规范中去。即使不知道这些扩展首部的含义，HTTP 程序也要接受它们并对其进行转发。

3.5.1　通用首部

有些首部提供了与报文相关的最基本的信息，它们被称为通用首部。它们像和事佬儿一样，不论报文是何类型，都为其提供一些有用信息。

例如，不管是构建请求报文还是响应报文，创建报文的日期和时间都是同一个意思，因此提供这类信息的首部对这两种类型的报文来说也是通用的。表 3-11 列出了通用的信息性首部。

表3-11　通用的信息性首部

首　　部	描　　述
`Connection`	允许客户端和服务器指定与请求 / 响应连接有关的选项
`Date`[5]	提供日期和时间标志，说明报文是什么时间创建的
`MIME-Version`	给出了发送端使用的 MIME 版本
`Trailer`	如果报文采用了分块传输编码（chunked transfer encoding）方式，就可以用这个首部列出位于报文拖挂（trailer）部分的首部集合[6]
`Transfer-Encoding`	告知接收端为了保证报文的可靠传输，对报文采用了什么编码方式
`Update`	给出了发送端可能想要“升级”使用的新版本或协议
`Via`	显示了报文经过的中间节点（代理、网关）

注 5：`Date` 中列出了 `Date` 首部可接受的日期格式。

注 6：15.6.3 节详细探讨了分块传输编码。

通用缓存首部

HTTP/1.0 引入了第一个允许 HTTP 应用程序缓存对象本地副本的首部，这样就不需要总是直接从源端服务器获取了。最新的 HTTP 版本有非常丰富的缓存参数集。第 7 章深入讨论了缓存。表 3-12 列出了基本的缓存首部。

表3-12 通用缓存首部

68

首　　部	描　　述
`Cache-Control`	用于随报文传送缓存指示
`Pragma`[7]	另一种随报文传送指示的方式，但并不专用于缓存

3.5.2 请求首部

请求首部是只在请求报文中有意义的首部。用于说明是谁或什么在发送请求、请求源自何处，或者客户端的喜好及能力。服务器可以根据请求首部给出的客户端信息，试着为客户端提供更好的响应。表 3-13 列出了请求的信息性首部。

表3-13 请求的信息性首部

首　　部	描　　述
`Client-IP`[8]	提供了运行客户端的机器的 IP 地址
`From`	提供了客户端用户的 E-mail 地址[9]
`Host`	给出了接收请求的服务器的主机名和端口号
`Referer`	提供了包含当前请求 URI 的文档的 URL
`UA-Color`	提供了与客户端显示器的显示颜色有关的信息
`UA-CPU`[10]	给出了客户端 CPU 的类型或制造商
`UA-Disp`	提供了与客户端显示器（屏幕）能力有关的信息
`UA-OS`	给出了运行在客户端机器上的操作系统名称及版本
`UA-Pixels`	提供了客户端显示器的像素信息
`User-Agent`	将发起请求的应用程序名称告知服务器

1. Accept首部

`Accept` 首部为客户端提供了一种将其喜好和能力告知服务器的方式，包括它们想要什么，可以使用什么，以及最重要的，它们不想要什么。这样，服务器就可以根

注 7：从技术角度来看，`Pragma` 是一种请求首部。从未被指定用于响应首部。由于经常被错误地用于响应首部，很多客户端和代理都会将 `Pragma` 解释为响应首部，但其确切语义并未得到很好地定义。任何情况下 `Cache-Control` 的使用都优于 `Pragma`。

注 8：RFC 2616 没有定义 `Client-IP` 和 `UA-*` 首部，但很多 HTTP 客户端应用程序都实现了这两个首部。

注 9：使用 RFC 822 E-mail 地址格式。

注 10：尽管有些客户端实现了 `UA-*` 首部，但我们认为 `UA-*` 首部是有副作用的。不应该将内容，尤其是 HTML，局限于特定的客户端配置。

据这些额外信息，对要发送的内容做出更明智的决定。Accept 首部会使连接的两端都受益。客户端会得到它们想要的内容，服务器则不会浪费其时间和带宽来发送客户端无法使用的东西。表 3-14 列出了各种 Accept 首部。

表3-14　Accept首部

首　　部	描　　述
Accept	告诉服务器能够发送哪些媒体类型
Accept-Charset	告诉服务器能够发送哪些字符集
Accept-Encoding	告诉服务器能够发送哪些编码方式
Accept-Language	告诉服务器能够发送哪些语言
TE[11]	告诉服务器可以使用哪些扩展传输编码

69

2. 条件请求首部

有时客户端希望为请求加上某些限制。比如，如果客户端已经有了一份文档副本，就希望只在服务器上的文档与客户端拥有的副本有所区别时，才请求服务器传输文档。通过条件请求首部，客户端就可以为请求加上这种限制，要求服务器在对请求进行响应之前，确保某个条件为真。表 3-15 列出了各种条件请求首部。

表3-15　条件请求首部

首　　部	描　　述
Expect	允许客户端列出某请求所要求的服务器行为
If-Match	如果实体标记与文档当前的实体标记相匹配，就获取这份文档[12]
If-Modified-Since	除非在某个指定的日期之后资源被修改过，否则就限制这个请求
If-None-Match	如果提供的实体标记与当前文档的实体标记不相符，就获取文档
If-Range	允许对文档的某个范围进行条件请求
If-Unmodified-Since	除非在某个指定日期之后资源没有被修改过，否则就限制这个请求
Range	如果服务器支持范围请求，就请求资源的指定范围[13]

3. 安全请求首部

HTTP 本身就支持一种简单的机制，可以对请求进行质询 / 响应认证。这种机制要求客户端在获取特定的资源之前，先对自身进行认证，这样就可以使事务稍微安全一些。我们会在第 14 章讨论这种质询 / 响应机制，同时还会对在 HTTP 之上实现的其他安全机制进行讨论。表 3-16 列出了一些安全请求首部。

注 11：更多有关 TE 首部的内容请参见 15.6.2 节。

注 12：更多有关实体标记的内容请参见第 7 章。标记本质上就是某版本资源的标识符。

注 13：更多有关 Range 首部的内容请参见 15.9 节。

表3-16 安全请求首部

首 部	描 述
Authorization	包含了客户端提供给服务器，以便对其自身进行认证的数据
Cookie	客户端用它向服务器传送一个令牌——它并不是真正的安全首部，但确实隐含了安全功能[14]
Cookie2	用来说明请求端支持的 cookie 版本，参见 11.6.7 节

4. 代理请求首部

随着因特网上代理的普遍应用，人们定义了几个首部来协助其更好地工作。第 6 章
70 对这些首部进行了详细的讨论。表 3-17 列出了一些代理请求首部。

表3-17 代理请求首部

首 部	描 述
Max-Forward	在通往源端服务器的路径上，将请求转发给其他代理或网关的最大次数——与 TRACE 方法一同使用[15]
Proxy-Authorization	与 Authorization 首部相同，但这个首部是在与代理进行认证时使用的
Proxy-Connection	与 Connection 首部相同，但这个首部是在与代理建立连接时使用的

3.5.3 响应首部

响应报文有自己的响应首部集。响应首部为客户端提供了一些额外信息，比如谁在发送响应、响应者的功能，甚至与响应相关的一些特殊指令。这些首部有助于客户端处理响应，并在将来发起更好的请求。表 3-18 列出了一些响应的信息性首部。

表3-18 响应的信息性首部

首 部	描 述
Age	（从最初创建开始）响应持续时间[16]
Public[17]	服务器为其资源支持的请求方法列表
Retry-After	如果资源不可用的话，在此日期或时间重试
Server	服务器应用程序软件的名称和版本
Title[18]	对 HTML 文档来说，就是 HTML 文档的源端给出的标题
Warning	比原因短语中更详细一些的警告报文

注 14：RFC 2616 并没有定义 Cookie 首部，在第 11 章详细讨论了该首部。

注 15：参见 6.6.2 节。

注 16：暗示响应是通过中间节点，很可能是从代理的缓存传送过来的。

注 17：Public 首部是在 RFC 2068 中定义的，但在最新的 HTTP 定义（RFC 2616）中并没有出现。

注 18：RFC 2616 并没有定义 Title 首部。请参见原始的 HTTP/1.0 草案定义（http://www.w3.org/Protocols/HTTP/HTTP2.html）。

1. 协商首部

如果资源有多种表示方法——比如，如果服务器上有某文档的法语和德语译稿，HTTP/1.1 可以为服务器和客户端提供对资源进行协商的能力。第 17 章详细讨论了协商。这里列出了几个首部，服务器可以用它们来传递与可协商资源有关的信息。表 3-19 列出了协商首部。

表3-19　协商首部

首　　部	描　　述
`Accept-Ranges`	对此资源来说，服务器可接受的范围类型
`Vary`	服务器查看的其他首部的列表，可能会使响应发生变化；也就是说，这是一个首部列表，服务器会根据这些首部的内容挑选出最适合的资源版本发送给客户端

71

2. 安全响应首部

我们已经看到过安全请求首部了，本质上这里说的就是 HTTP 的质询 / 响应认证机制的响应侧。我们会在第 14 章对安全问题进行详细的讨论。现在这里介绍的是一些基本的质询首部。表 3-20 列出了安全响应首部。

表3-20　安全响应首部

首　　部	描　　述
`Proxy-Authenticate`	来自代理的对客户端的质询列表
`Set-Cookie`	不是真正的安全首部，但隐含有安全功能；可以在客户端设置一个令牌，以便服务器对客户端进行标识[19]
`Set-Cookie2`	与 `Set-Cookie` 类似，RFC 2965 Cookie 定义；参见 11.6.7 节
`WWW-Authenticate`	来自服务器的对客户端的质询列表

3.5.4　实体首部

有很多首部可以用来描述 HTTP 报文的负荷。由于请求和响应报文中都可能包含实体部分，所以在这两种类型的报文中都可能出现这些首部。

实体首部提供了有关实体及其内容的大量信息，从有关对象类型的信息，到能够对资源使用的各种有效的请求方法。总之，实体首部可以告知报文的接收者它在对什么进行处理。表 3-21 列出了实体的信息性首部。

注 19：`Set-Cookie` 和 `Set-Cookie2` 都是扩展首部，参见第 11 章。

表3-21　实体的信息性首部

首　　部	描　　述
Allow	列出了可以对此实体执行的请求方法
Location	告知客户端实体实际上位于何处；用于将接收端定向到资源的（可能是新的）位置（URL）上去

1. 内容首部

内容首部提供了与实体内容有关的特定信息，说明了其类型、尺寸以及处理它所需的其他有用信息。比如，Web 浏览器可以通过查看返回的内容类型，得知如何显示对象。表 3-22 列出了各种内容首部。

表3-22　内容首部

首　　部	描　　述
Content-Base[20]	解析主体中的相对 URL 时使用的基础 URL
Content-Encoding	对主体执行的任意编码方式
Content-Language	理解主体时最适宜使用的自然语言
Content-Length	主体的长度或尺寸
Content-Location	资源实际所处的位置
Content-MD5	主体的 MD5 校验和
Content-Range	在整个资源中此实体表示的字节范围
Content-Type	这个主体的对象类型

72

2. 实体缓存首部

通用的缓存首部说明了如何或什么时候进行缓存。实体的缓存首部提供了与被缓存实体有关的信息——比如，验证已缓存的资源副本是否仍然有效所需的信息，以及更好地估计已缓存资源何时失效所需的线索。

第 7 章深入讨论了 HTTP 请求和响应的缓存。在那里我们会再次看到这些首部。表 3-23 列出了一些实体缓存首部。

表3-23　实体缓存首部

首　　部	描　　述
ETag	与此实体相关的实体标记[21]
Expires	实体不再有效，要从原始的源端再次获取此实体的日期和时间
Last-Modified	这个实体最后一次被修改的日期和时间

注 20：RFC 2616 中没有定义 Content-Base 首部。

注 21：实体标记本质上来说就是某个特定资源版本的标识符。

3.6 更多信息

更多信息请参见下列资源。

- http://www.w3.org/Protocols/rfc2616/rfc2616.txt
  RFC 2616，“Hypertext Transfer Protocol”，由 R. Fielding、J. Gettys、J. Mogul、H. Frystyk、L. Mastinter、P. Leach 和 T. Berners-Lee 编写。

- *HTTP Pocket Reference*（《HTTP 口袋书》）
  Clintin Wong 编写，O’Reilly & Associates 公司出版。

- http://www.w3.org/Protocols/
  HTTP 的 W3C 架构页面。 73

第4章

连接管理

HTTP 规范对 HTTP 报文解释得很清楚，但对 HTTP 连接介绍的并不多，HTTP 连接是 HTTP 报文传输的关键通道。编写 HTTP 应用程序的程序员需要理解 HTTP 连接的来龙去脉以及如何使用这些连接。

HTTP 连接管理有点像魔术，应当从经验与实践，而不仅仅是出版的文献中学习。通过本章，可以了解到：

- HTTP 是如何使用 TCP 连接的；
- TCP 连接的时延、瓶颈以及存在的障碍；
- HTTP 的优化，包括并行连接、keep-alive（持久连接）和管道化连接；
- 管理连接时应该以及不应该做的事情。

4.1 TCP连接

世界上几乎所有的 HTTP 通信都是由 TCP/IP 承载的，TCP/IP 是全球计算机及网络设备都在使用的一种常用的分组交换网络分层协议集。客户端应用程序可以打开一条 TCP/IP 连接，连接到可能运行在世界任何地方的服务器应用程序。一旦连接建立起来了，在客户端和服务器的计算机之间交换的报文就永远不会丢失、受损或失序。[1]

比如，你想获取 Joe 的五金商店最新的电动工具价目表：

http://www.joes-hardware.com:80/power-tools.html

浏览器收到这个 URL 时，会执行图 4-1 所示的步骤。第 (1) ～ (3) 步会将服务器的
74 IP 地址和端口号从 URL 中分离出来。在第 (4) 步中建立到 Web 服务器的 TCP 连接，并在第 (5) 步通过这条连接发送一条请求报文。在第 (6) 步读取响应，并在第 (7) 步关闭连接。

4.1.1 TCP的可靠数据管道

HTTP 连接实际上就是 TCP 连接及其使用规则。TCP 连接是因特网上的可靠连接。要想正确、快速地发送数据，就需要了解 TCP 的一些基本知识。[2]

注 1：尽管报文不会丢失或受损，但如果计算机或网络崩溃了，客户端和服务器之间的通信仍然会被断开。在这种情况下，会通知客户端和服务器通信中断了。

注 2：如果要编写复杂的 HTTP 应用程序，尤其是，希望程序能够快速运行的话，所需学习的、与 TCP 内部原理及性能有关的知识就要比本章所讨论的内容多得多。我们推荐 W. Richard Stevens 编写的 *TCP/IP Illustrated*（《TCP/IP 详解》）系列图书（Addison Wesley 公司出版）。

TCP 为 HTTP 提供了一条可靠的比特传输管道。从 TCP 连接一端填入的字节会从另 75
一端以原有的顺序、正确地传送出来（参见图 4-2）。

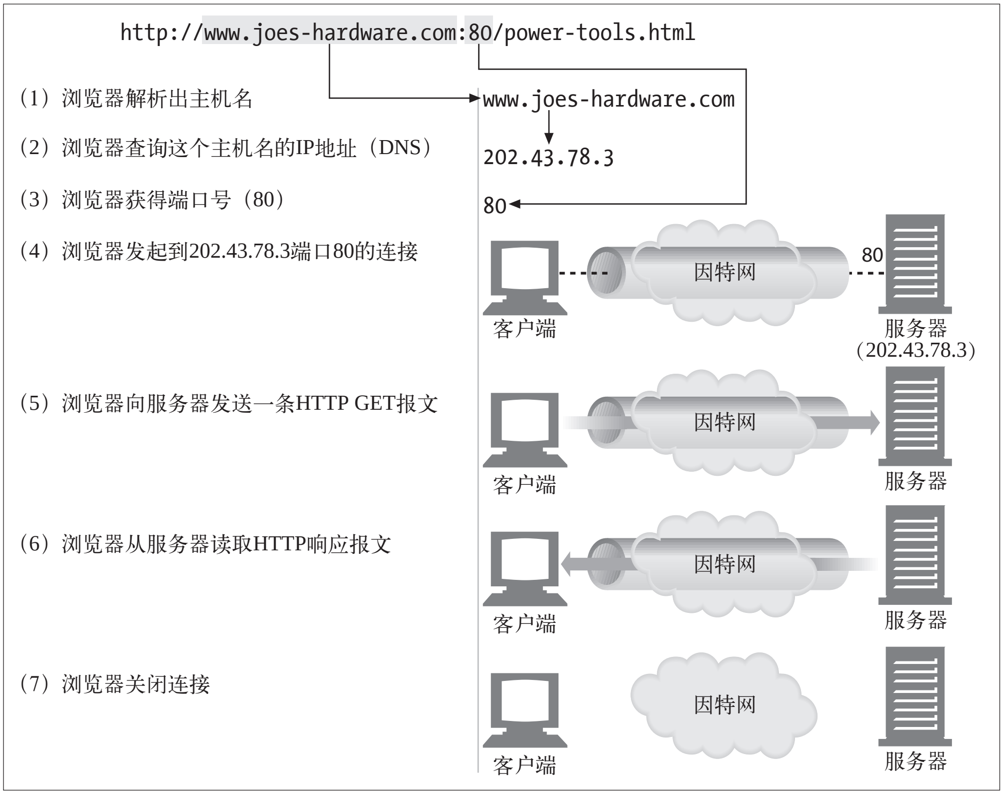


图 4-1　Web 浏览器通过 TCP 连接与 Web 服务器进行交互



图 4-2　TCP 会按序、无差错地承载 HTTP 数据

4.1.2　TCP流是分段的、由IP分组传送

TCP 的数据是通过名为 IP 分组（或 IP 数据报）的小数据块来发送的。这样的话，如图 4-3a 所示，HTTP 就是“HTTP over TCP over IP”这个“协议栈”中的最顶层了。其安全版本 HTTPS 就是在 HTTP 和 TCP 之间插入了一个（称为 TLS 或 SSL 的）密码加密层（图 4-3b）。

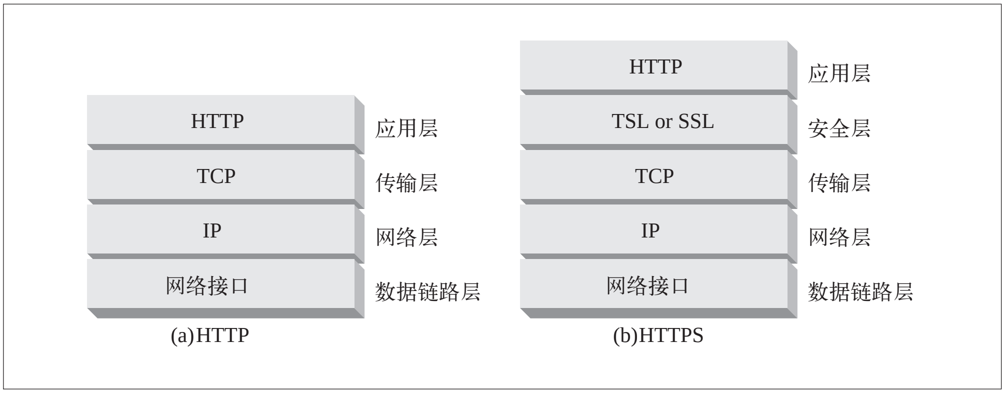


图 4-3　HTTP 和 HTTPS 网络协议栈

HTTP 要传送一条报文时，会以流的形式将报文数据的内容通过一条打开的 TCP 连接按序传输。TCP 收到数据流之后，会将数据流砍成被称作段的小数据块，并将段封装在 IP 分组中，通过因特网进行传输（参见图 4-4）。所有这些工作都是由 TCP/IP 软件来处理的，HTTP 程序员什么都看不到。

每个 TCP 段都是由 IP 分组承载，从一个 IP 地址发送到另一个 IP 地址的。每个 IP 分组中都包括：

- 一个 IP 分组首部（通常为 20 字节）；
- 一个 TCP 段首部（通常为 20 字节）；
- 一个 TCP 数据块（0 个或多个字节）。

IP 首部包含了源和目的 IP 地址、长度和其他一些标记。TCP 段的首部包含了 TCP
76 端口号、TCP 控制标记，以及用于数据排序和完整性检查的一些数字值。

4.1.3　保持TCP连接持续不断地运行

在任意时刻计算机都可以有几条 TCP 连接处于打开状态。TCP 是通过*端口号*来保持所有这些连接持续不断地运行。

端口号和雇员使用的电话分机号很类似。就像公司的总机号码能将你接到前台，而分机号可以将你接到正确的雇员位置一样，IP 地址可以将你连接到正确的计算机，而端口号则可以将你连接到正确的应用程序上去。TCP 连接是通过 4 个值来识别的：

< 源 IP 地址、源端口号、目的 IP 地址、目的端口号 >

77 这 4 个值一起唯一地定义了一条连接。两条不同的 TCP 连接不能拥有 4 个完全相同的地址组件值（但不同连接的部分组件可以拥有相同的值）。

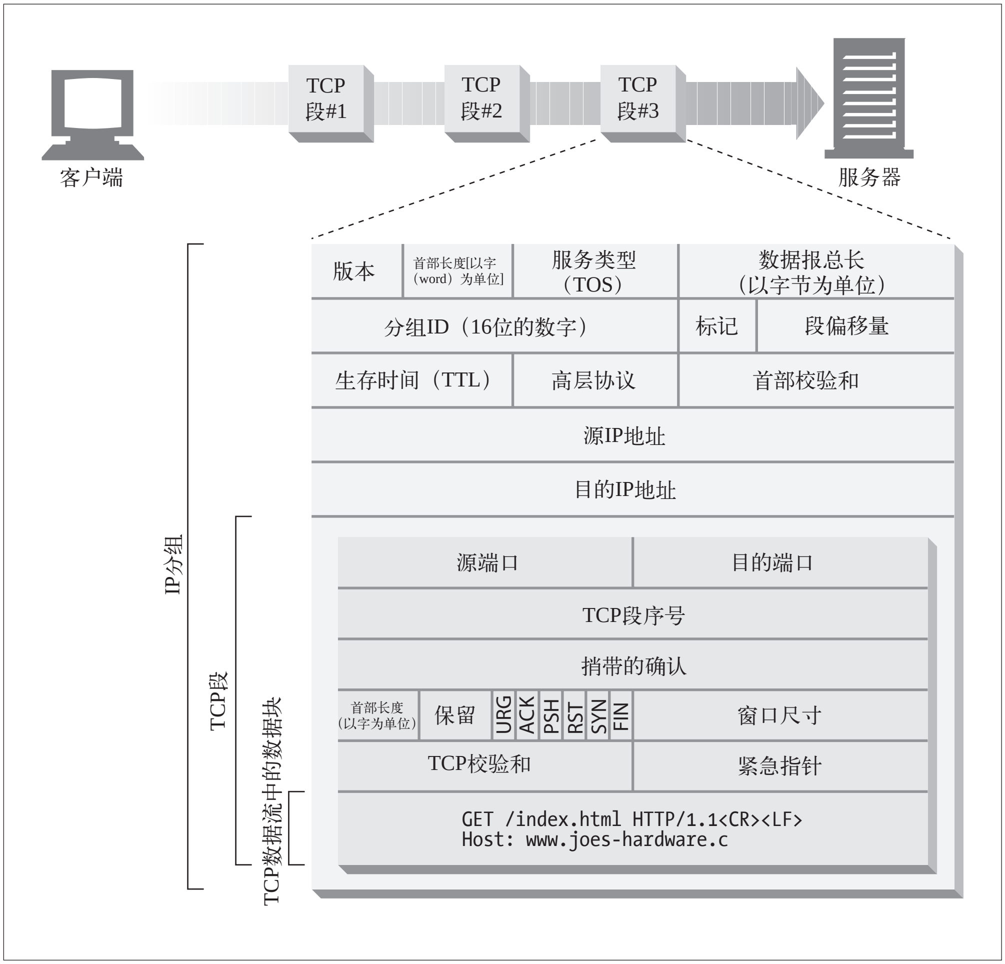


图 4-4　承载 TCP 段的 IP 分组，它承载了 TCP 数据流中的小块数据

在图 4-5 中，有 4 条连接：A、B、C 和 D。表 4-1 列出了每个端口的相关信息。

表4-1　TCP连接值

连　　接	源IP地址	源　端　口	目的IP地址	目的端口
A	209.1.32.34	2034	204.62.128.58	4133
B	209.1.32.35	3227	204.62.128.58	4140
C	209.1.32.35	3105	207.25.71.25	80
D	209.1.33.89	5100	207.25.71.25	80

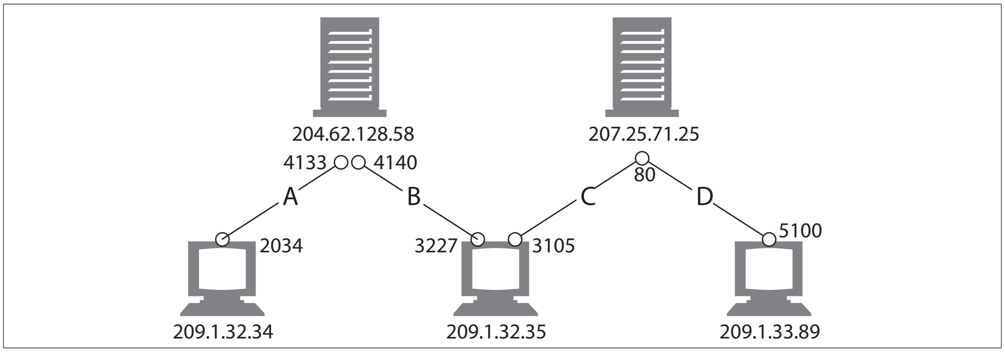


图 4-5　4 个不同的 TCP 连接

注意，有些连接共享了相同的目的端口号（C 和 D 都使用目的端口号 80）。有些连接使用了相同的源 IP 地址（B 和 C）。有些使用了相同的目的 IP 地址（A 和 B，C 和 D）。但没有两个不同连接所有的 4 个值都一样。

4.1.4　用TCP套接字编程

操作系统提供了一些操纵其 TCP 连接的工具。为了更具体地说明问题，我们来看一个 TCP 编程接口。表 4-2 显示了套接字 API 提供的一些主要接口。这个套接字 API 向 HTTP 程序员隐藏了 TCP 和 IP 的所有细节。套接字 API 最初是为 Unix 操作系统开发的，但现在几乎所有的操作系统和语言中都有其变体存在。

表4-2　对TCP连接进行编程所需的常见套接字接口函数

套接字API调用	描　述
s = socket(<parameters>)	创建一个新的、未命名、未关联的套接字
78 bind(s,<local IP:port>)	向套接字赋一个本地端口号和接口
connect(s, <remote IP:port>)	创建一条连接本地套接字与远程主机及端口的连接
listen(s,...)	标识一个本地套接字，使其可以合法接受连接
s2 = accept(s)	等待某人建立一条到本地端口的连接
n = read(s, buffer, n)	尝试从套接字向缓冲区读取 *n* 个字节
n = write(s, buffer, n)	尝试从缓冲区中向套接字写入 *n* 个字节
close(s)	完全关闭 TCP 连接
shutdown(s,<side>)	只关闭 TCP 连接的输入或输出端
getsockopt(s,...)	读取某个内部套接字配置选项的值
setsockopt(s,...)	修改某个内部套接字配置选项的值

套接字 API 允许用户创建 TCP 的端点数据结构，将这些端点与远程服务器的 TCP 端点进行连接，并对数据流进行读写。TCP API 隐藏了所有底层网络协议的握手细节，以及 TCP 数据流与 IP 分组之间的分段和重装细节。

图 4-1 显示了 Web 浏览器是如何用 HTTP 从 Joe 的五金商店下载 power-tools.html 页面的。图 4-6 中的伪代码说明了可以怎样通过套接字 API 来凸显客户端和服务器在实现 HTTP 事务时所应执行的步骤。

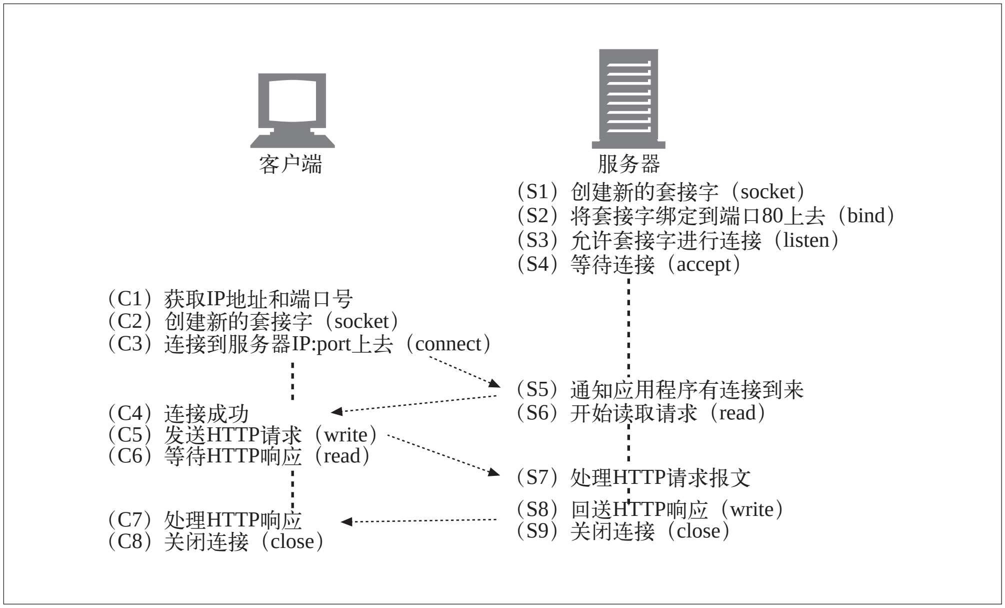


图 4-6　TCP 客户端和服务器是如何通过 TCP 套接字接口进行通信的 79

我们从 Web 服务器等待连接（参见图 4-6，S4）开始。客户端根据 URL 判定出 IP 地址和端口号，并建立一条到服务器的 TCP 连接（参见图 4-6，C3）。建立连接可能要花费一些时间，时间长短取决于服务器距离的远近、服务器的负载情况，以及因特网的拥挤程度。

一旦建立了连接，客户端就会发送 HTTP 请求（参见图 4-6，C5），服务器则会读取请求（参见图 4-6，S6）。一旦服务器获取了整条请求报文，就会对请求进行处理，执行所请求的动作（参见图 4-6，S7），并将数据写回客户端。客户端读取数据（参见图 4-6，C6），并对响应数据进行处理（参见图 4-6，C7）。

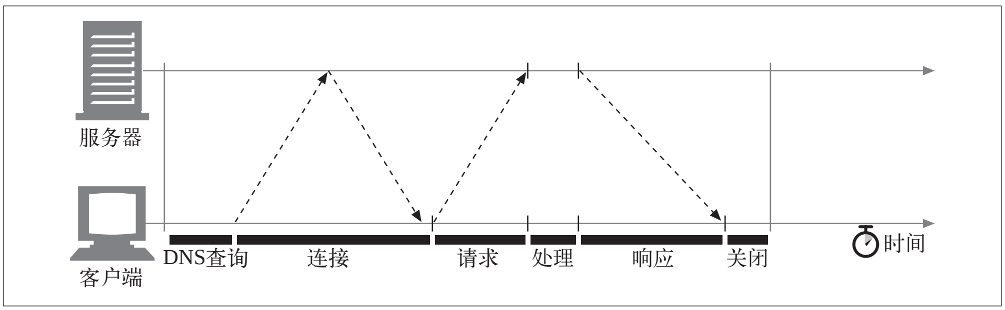
4.2　对TCP性能的考虑

HTTP 紧挨着 TCP，位于其上层，所以 HTTP 事务的性能在很大程度上取决于底层 TCP 通道的性能。本节重点介绍了一些很重要的、对这些 TCP 连接的性能考虑。理解了 TCP 的某些基本性能特点之后，就可以更好地理解 HTTP 的连接优化特性，这样就能设计实现一些更高性能的 HTTP 应用程序了。

本节要求大家对 TCP 协议的内部细节有一定的了解。如果对 TCP 性能考虑的细节不感兴趣（或者很熟悉这些细节），可以直接跳到 4.3 节。TCP 是个很复杂的话题，所以这里我们只能提供对 TCP 性能的简要概述。本章末尾的 4.8 节列出了一些很好的 TCP 参考书，以供参考。

4.2.1 HTTP事务的时延

我们来回顾一下，在 HTTP 请求的过程中会出现哪些网络时延，并以此开始我们的 TCP 性能之旅。图 4-7 描绘了 HTTP 事务主要的连接、传输以及处理时延。



80 图 4-7 串行 HTTP 事务的时间线

注意，与建立 TCP 连接，以及传输请求和响应报文的时间相比，事务处理时间可能是很短的。除非客户端或服务器超载，或正在处理复杂的动态资源，否则 HTTP 时延就是由 TCP 网络时延构成的。

HTTP 事务的时延有以下几种主要原因。

(1) 客户端首先需要根据 URI 确定 Web 服务器的 IP 地址和端口号。如果最近没有对 URI 中的主机名进行访问，通过 DNS 解析系统将 URI 中的主机名转换成一个 IP 地址可能要花费数十秒的时间[3]。

(2) 接下来，客户端会向服务器发送一条 TCP 连接请求，并等待服务器回送一个请求接受应答。每条新的 TCP 连接都会有连接建立时延。这个值通常最多只有一两秒钟，但如果有数百个 HTTP 事务的话，这个值会快速地叠加上去。

(3) 一旦连接建立起来了，客户端就会通过新建立的 TCP 管道来发送 HTTP 请求。数据到达时，Web 服务器会从 TCP 连接中读取请求报文，并对请求进行处理。

注 3：幸运的是，大多数 HTTP 客户端都有一个小的 DNS 缓存，用来保存近期所访问站点的 IP 地址。如果已经在本地“缓存”（记录）了 IP 地址，查询就可以立即完成。因为大多数 Web 浏览器浏览的都是少数常用站点，所以通常都可以很快地将主机名解析出来。

因特网传输请求报文，以及服务器处理请求报文都需要时间。

(4) 然后，Web 服务器会回送 HTTP 响应，这也需要花费时间。

这些 TCP 网络时延的大小取决于硬件速度、网络和服务器的负载，请求和响应报文的尺寸，以及客户端和服务器之间的距离。TCP 协议的技术复杂性也会对时延产生巨大的影响。

4.2.2　性能聚焦区域

本节其余部分列出了一些会对 HTTP 程序员产生影响的、最常见的 TCP 相关时延，其中包括：

- TCP 连接建立握手；
- TCP 慢启动拥塞控制；
- 数据聚集的 Nagle 算法；
- 用于捎带确认的 TCP 延迟确认算法；
- TIME_WAIT 时延和端口耗尽。

如果要编写高性能的 HTTP 软件，就应该理解上面的每一个因素。如果不需要进行这个级别的性能优化，可以跳过这部分内容。 81

4.2.3　TCP连接的握手时延

建立一条新的 TCP 连接时，甚至是在发送任意数据之前，TCP 软件之间会交换一系列的 IP 分组，对连接的有关参数进行沟通（参见图 4-8）。如果连接只用来传送少量数据，这些交换过程就会严重降低 HTTP 的性能。

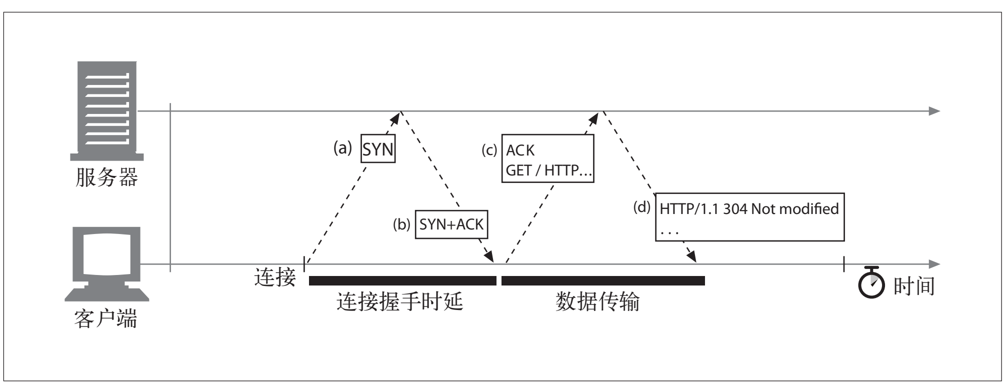


图 4-8　在发送数据之前，TCP 要传送两个分组来建立连接

TCP 连接握手需要经过以下几个步骤。

(1) 请求新的 TCP 连接时，客户端要向服务器发送一个小的 TCP 分组（通常是 40 ~ 60 个字节）。这个分组中设置了一个特殊的 SYN 标记，说明这是一个连接请求。（参见图 4-8a）。

(2) 如果服务器接受了连接，就会对一些连接参数进行计算，并向客户端回送一个 TCP 分组，这个分组中的 SYN 和 ACK 标记都被置位，说明连接请求已被接受（参见图 4-8b）。

(3) 最后，客户端向服务器回送一条确认信息，通知它连接已成功建立（参见图 4-8c）。现代的 TCP 栈都允许客户端在这个确认分组中发送数据。

HTTP 程序员永远不会看到这些分组——这些分组都由 TCP/IP 软件管理，对其是不可见的。HTTP 程序员看到的只是创建 TCP 连接时存在的时延。

通常 HTTP 事务都不会交换太多数据，此时，SYN/SYN+ACK 握手（参见图 4-8a 和图 4-8b）会产生一个可测量的时延。TCP 连接的 ACK 分组（参见图 4-8c）通常都足够大，可以承载整个 HTTP 请求报文[4]，而且很多 HTTP 服务器响应报文都可以放入一个 IP 分组中去（比如，响应是包含了装饰性图片的小型 HTML 文件，或
82 者是对浏览器高速缓存请求产生的 304 Not Modified 响应）。

最后的结果是，小的 HTTP 事务可能会在 TCP 建立上花费 50%，或更多的时间。后面的小节会讨论 HTTP 是如何通过重用现存连接，来减小这种 TCP 建立时延所造成的影响的。

4.2.4　延迟确认

由于因特网自身无法确保可靠的分组传输（因特网路由器超负荷的话，可以随意丢弃分组），所以 TCP 实现了自己的确认机制来确保数据的成功传输。

每个 TCP 段都有一个序列号和数据完整性校验和。每个段的接收者收到完好的段时，都会向发送者回送小的确认分组。如果发送者没有在指定的窗口时间内收到确认信息，发送者就认为分组已被破坏或损毁，并重发数据。

由于确认报文很小，所以 TCP 允许在发往相同方向的输出数据分组中对其进行“捎带”。TCP 将返回的确认信息与输出的数据分组结合在一起，可以更有效地利用网络。为了增加确认报文找到同向传输数据分组的可能性，很多 TCP 栈都实现了一种

注 4：因特网流量中的 IP 分组通常是几百字节，本地流量中的 IP 分组为 1500 字节左右。

“延迟确认”算法。延迟确认算法会在一个特定的窗口时间（通常是 100 ～ 200 毫秒）内将输出确认存放在缓冲区中，以寻找能够捎带它的输出数据分组。如果在那个时间段内没有输出数据分组，就将确认信息放在单独的分组中传送。

但是，HTTP 具有双峰特征的请求 – 应答行为降低了捎带信息的可能。当希望有相反方向回传分组的时候，偏偏没有那么多。通常，延迟确认算法会引入相当大的时延。根据所使用操作系统的不同，可以调整或禁止延迟确认算法。

在对 TCP 栈的任何参数进行修改之前，一定要对自己在做什么有清醒的认识。TCP 中引入这些算法的目的是防止设计欠佳的应用程序对因特网造成破坏。对 TCP 配置进行的任意修改，都要绝对确保应用程序不会引发这些算法所要避免的问题。

4.2.5　TCP慢启动

TCP 数据传输的性能还取决于 TCP 连接的*使用期*（age）。TCP 连接会随着时间进行自我“调谐”，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度。这种调谐被称为 TCP *慢启动*（slow start），用于防止因特网的突然过载和拥塞。 83

TCP 慢启动限制了一个 TCP 端点在任意时刻可以传输的分组数。简单来说，每成功接收一个分组，发送端就有了发送另外两个分组的权限。如果某个 HTTP 事务有大量数据要发送，是不能一次将所有分组都发送出去的。必须发送一个分组，等待确认；然后可以发送两个分组，每个分组都必须被确认，这样就可以发送四个分组了，以此类推。这种方式被称为“打开拥塞窗口”。

由于存在这种拥塞控制特性，所以新连接的传输速度会比已经交换过一定量数据的、“已调谐”连接慢一些。由于已调谐连接要更快一些，所以 HTTP 中有一些可以重用现存连接的工具。本章稍后会介绍这些 HTTP“持久连接”。

4.2.6　Nagle算法与TCP_NODELAY

TCP 有一个数据流接口，应用程序可以通过它将任意尺寸的数据放入 TCP 栈中——即使一次只放一个字节也可以！但是，每个 TCP 段中都至少装载了 40 个字节的标记和首部，所以如果 TCP 发送了大量包含少量数据的分组，网络的性能就会严重下降。[5]

注 5：发送大量单字节分组的行为称为“发送端傻窗口综合症”。这种行为效率很低、违反社会道德，而且可能会影响其他的因特网流量。

Nagle 算法（根据其发明者 John Nagle 命名）试图在发送一个分组之前，将大量 TCP 数据绑定在一起，以提高网络效率。RFC 896“IP/TCP 互连网络中的拥塞控制”对此算法进行了描述。

Nagle 算法鼓励发送全尺寸（LAN 上最大尺寸的分组大约是 1500 字节，在因特网上是几百字节）的段。只有当所有其他分组都被确认之后，Nagle 算法才允许发送非全尺寸的分组。如果其他分组仍然在传输过程中，就将那部分数据缓存起来。只有当挂起分组被确认，或者缓存中积累了足够发送一个全尺寸分组的数据时，才会将缓存的数据发送出去。[6]

Nagle 算法会引发几种 HTTP 性能问题。首先，小的 HTTP 报文可能无法填满一个分组，可能会因为等待那些永远不会到来的额外数据而产生时延。其次，Nagle 算法与延迟确认之间的交互存在问题——Nagle 算法会阻止数据的发送，直到有确认分组抵达为止，但确认分组自身会被延迟确认算法延迟 100 ～ 200 毫秒。[7]

HTTP 应用程序常常会在自己的栈中设置参数 TCP_NODELAY，禁用 Nagle 算法，提高性能。如果要这么做的话，一定要确保会向 TCP 写入大块的数据，这样就不会
84 产生一堆小分组了。

4.2.7　TIME_WAIT累积与端口耗尽

TIME_WAIT 端口耗尽是很严重的性能问题，会影响到性能基准，但在现实中相对较少出现。大多数遇到性能基准问题的人最终都会碰到这个问题，而且性能都会变得出乎意料地差，所以这个问题值得特别关注。

当某个 TCP 端点关闭 TCP 连接时，会在内存中维护一个小的控制块，用来记录最近所关闭连接的 IP 地址和端口号。这类信息只会维持一小段时间，通常是所估计的最大分段使用期的两倍（称为 2MSL，通常为 2 分钟 [8]）左右，以确保在这段时间内不会创建具有相同地址和端口号的新连接。实际上，这个算法可以防止在两分钟内创建、关闭并重新创建两个具有相同 IP 地址和端口号的连接。

现在高速路由器的使用，使得重复分组几乎不可能在连接关闭的几分钟之后，出现在服务器上。有些操作系统会将 2MSL 设置为一个较小的值，但修改此值时要特别

注 6：这个算法有几种变体，包括对超时和确认逻辑的修改，但基本算法会使数据的缓存比一个 TCP 段小一些。

注 7：使用管道化连接（本章稍后介绍）时这些问题可能会更加严重，因为客户端可能会有多条报文要发送给同一个服务器，而且不希望有时延存在。

注 8：将 2MSL 的值取为 2 分钟是有历史原因的。很早以前，路由器的速度还很慢，人们估计，在将一个分组的复制副本丢弃之前，它可以在因特网队列中保留最多一分钟的时间。现在，最大分段生存期要小得多了。

小心。分组确实会被复制，如果来自之前连接的复制分组插入了具有相同连接值的新 TCP 流，会破坏 TCP 数据。

2MSL 的连接关闭延迟通常不是什么问题，但在性能基准环境下就可能会成为一个问题。进行性能基准测试时，通常只有一台或几台用来产生流量的计算机连接到某系统中去，这样就限制了连接到服务器的客户端 IP 地址数。而且，服务器通常会在 HTTP 的默认 TCP 端口 80 上进行监听。用 TIME_WAIT 防止端口号重用时，这些情况也限制了可用的连接值组合。

在只有一个客户端和一台 Web 服务器的异常情况下，构建一条 TCP 连接的 4 个值：

```
<source-IP-address, source-port, destination-IP-address, destination-port>
```

其中的 3 个都是固定的——只有源端口号可以随意改变：

```
<client-IP, source-port, server-IP, 80>
```

客户端每次连接到服务器上去时，都会获得一个新的源端口，以实现连接的唯一性。但由于可用源端口的数量有限（比如，60 000 个），而且在 2MSL 秒（比如，120 秒）内连接是无法重用的，连接率就被限制在了 60 000/120=500 次 / 秒。如果再不 85
断进行优化，并且服务器的连接率不高于 500 次 / 秒，就可确保不会遇到 TIME_WAIT 端口耗尽问题。要修正这个问题，可以增加客户端负载生成机器的数量，或者确保客户端和服务器在循环使用几个虚拟 IP 地址以增加更多的连接组合。

即使没有遇到端口耗尽问题，也要特别小心有大量连接处于打开状态的情况，或为处于等待状态的连接分配了大量控制块的情况。在有大量打开连接或控制块的情况下，有些操作系统的速度会严重减缓。

4.3　HTTP连接的处理

本章的前两节对 TCP 连接及其性能含义进行了精要的介绍。要想学习更多与 TCP 联网有关的知识，请参见本章末尾的资源列表。

现在我们切回到 HTTP 上来。本章其余部分将解释操作和优化连接的 HTTP 技术。我们从 HTTP 的 Connection 首部开始介绍，这是 HTTP 连接管理中一个很容易被误解，但又很重要的部分。然后会介绍一些 HTTP 连接优化技术。

4.3.1　常被误解的Connection首部

HTTP 允许在客户端和最终的源端服务器之间存在一串 HTTP 中间实体（代理、高速缓存等）。可以从客户端开始，逐跳地将 HTTP 报文经过这些中间设备，转发到源

端服务器上去（或者进行反向传输）。

在某些情况下，两个相邻的 HTTP 应用程序会为它们共享的连接应用一组选项。HTTP 的 Connection 首部字段中有一个由逗号分隔的连接标签列表，这些标签为此连接指定了一些不会传播到其他连接中去的选项。比如，可以用 Connection: close 来说明发送完下一条报文之后必须关闭的连接。

Connection 首部可以承载 3 种不同类型的标签，因此有时会很令人费解：

- HTTP 首部字段名，列出了只与此连接有关的首部；
- 任意标签值，用于描述此连接的非标准选项；
- 值 close，说明操作完成之后需关闭这条持久连接。

如果连接标签中包含了一个 HTTP 首部字段的名称，那么这个首部字段就包含了与一些连接有关的信息，不能将其转发出去。在将报文转发出去之前，必须删除
86 Connection 首部列出的所有首部字段。由于 Connection 首部可以防止无意中对本地首部的转发，因此将逐跳首部名放入 Connection 首部被称为“对首部的保护”。图 4-9 显示了一个这样的例子。

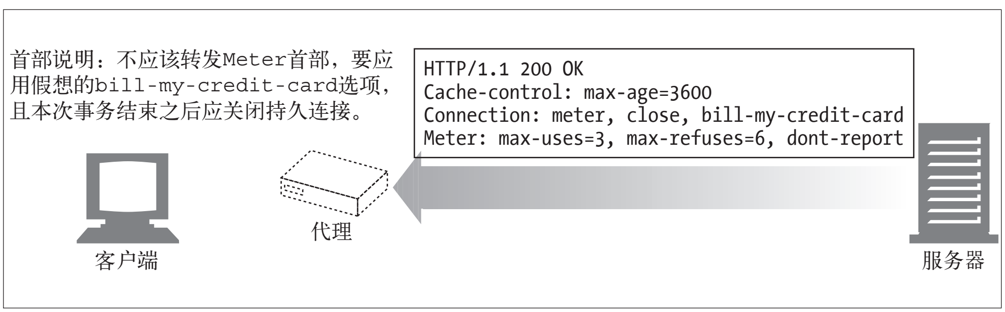


图 4-9 **Connection** 首部允许发送端指定与连接有关的选项

HTTP 应用程序收到一条带有 Connection 首部的报文时，接收端会解析发送端请求的所有选项，并将其应用。然后会在将此报文转发给下一跳地址之前，删除 Connection 首部以及 Connection 中列出的所有首部。而且，可能还会有少量没有作为 Connection 首部值列出，但一定不能被代理转发的逐跳首部。其中包括 Prxoy-Authenticate、Proxy-Connection、Transfer-Encoding 和 Upgrade。更多有关 Connection 首部的内容请参见附录 C。

4.3.2 串行事务处理时延

如果只对连接进行简单的管理，TCP 的性能时延可能会叠加起来。比如，假设有一

个包含了 3 个嵌入图片的 Web 页面。浏览器需要发起 4 个 HTTP 事务来显示此页面：1 个用于顶层的 HTML 页面，3 个用于嵌入的图片。如果每个事务都需要（串行地建立）一条新的连接，那么连接时延和慢启动时延就会叠加起来（参见图 4-10）。[9]

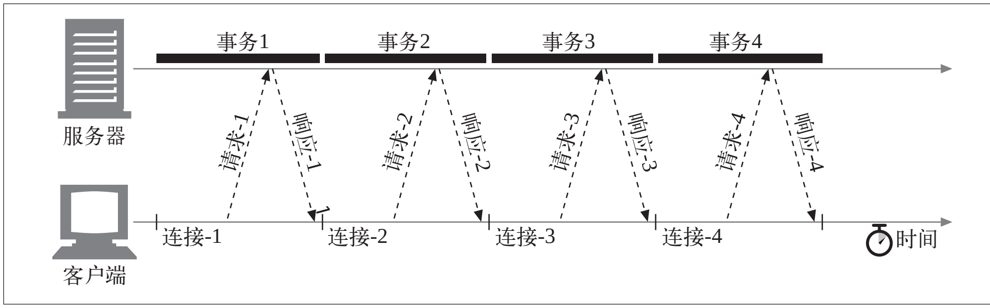


图 4-10　4 个事务（串行） 87

除了串行加载引入的实际时延之外，加载一幅图片时，页面上其他地方都没有动静也会让人觉得速度很慢。用户更希望能够同时加载多幅图片。[10]

串行加载的另一个缺点是，有些浏览器在对象加载完毕之前无法获知对象的尺寸，而且它们可能需要尺寸信息来决定将对象放在屏幕的什么位置上，所以在加载了足够多的对象之前，无法在屏幕上显示任何内容。在这种情况下，可能浏览器串行装载对象的进度很正常，但用户面对的却是一个空白的屏幕，对装载的进度一无所知。[11]

还有几种现存和新兴的方法可以提高 HTTP 的连接性能。后面几节讨论了四种此类技术。

- 并行连接
  通过多条 TCP 连接发起并发的 HTTP 请求。
- 持久连接
  重用 TCP 连接，以消除连接及关闭时延。
- 管道化连接
  通过共享的 TCP 连接发起并发的 HTTP 请求。

注 9：根据举此例的目的，假设所有对象的长度基本上都一样，并且是从同一台服务器发出的，而且 DNS 条目被缓存了，排除了 DNS 的查找时间。

注 10：即使同时加载多幅图片比一次加载一幅图片要慢，人们也会有同样的感觉！用户通常会认为多幅图片同时加载要快一些。

注 11：HTML 的设计者可以在图片等嵌入式对象的 HTML 标签中显式地添加宽高属性，以消除这种“布局时延”。显式地提供了嵌入图片的宽度和高度，浏览器就可以在从服务器收到对象之前确定图形的布局了。

- 复用的连接

  交替传送请求和响应报文（实验阶段）。

4.4 并行连接

如前所述，浏览器可以先完整地请求原始的 HTML 页面，然后请求第一个嵌入对象，然后请求第二个嵌入对象等，以这种简单的方式对每个嵌入式对象进行串行处理。但这样实在是太慢了！

如图 4-11 所示，HTTP 允许客户端打开多条连接，并行地执行多个 HTTP 事务。在这个例子中，并行加载了四幅嵌入式图片，每个事务都有自己的 TCP 连接。[12]

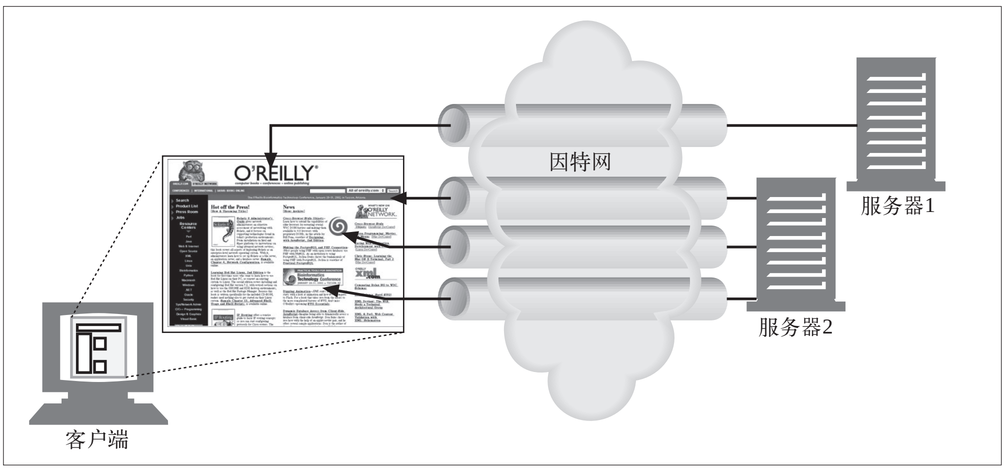


图 4-11　页面上的每个组件都包含一个独立的 HTTP 事务

4.4.1 并行连接可能会提高页面的加载速度

包含嵌入对象的组合页面如果能（通过并行连接）克服单条连接的空载时间和带宽
88 限制，加载速度也会有所提高。时延可以重叠起来，而且如果单条连接没有充分利用客户端的因特网带宽，可以将未用带宽分配来装载其他对象。

图 4-12 显示了并行连接的时间线，比图 4-10 要快得多。首先装载的是封闭的 HTML 页面，然后并行处理其余的 3 个事务，每个事务都有自己的连接。[13] 图片的装载是并行的，连接的时延也是重叠的。

注 12：嵌入的组件不一定都在同一台 Web 服务器上，可以同多台服务器建立并行的连接。

注 13：由于软件开销的存在，每个连接请求之间总是会有一些小的时延，但连接请求和传输时间基本上都是重叠起来的。

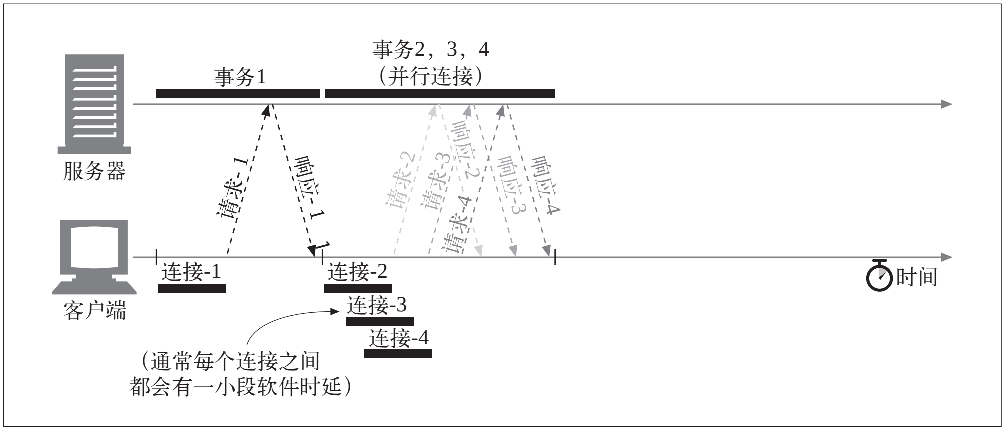


图 4-12　4 个事务（并行）

4.4.2　并行连接不一定更快

即使并行连接的速度可能会更快，但并不一定总是更快。客户端的网络带宽不足（比如，浏览器是通过一个 28.8kbps 的 Modem 连接到因特网上去的）时，大部分的时间可能都是用来传送数据的。在这种情况下，一个连接到速度较快服务器上的 HTTP 事务就会很容易地耗尽所有可用的 Modem 带宽。如果并行加载多个对象，每个对象都会去竞争这有限的带宽，每个对象都会以较慢的速度按比例加载，这样带来的性能提升就很小，甚至没什么提升。[14] 89

而且，打开大量连接会消耗很多内存资源，从而引发自身的性能问题。复杂的 Web 页面可能会有数十或数百个内嵌对象。客户端可能可以打开数百个连接，但 Web 服务器通常要同时处理很多其他用户的请求，所以很少有 Web 服务器希望出现这样的情况。一百个用户同时发出申请，每个用户打开 100 个连接，服务器就要负责处理 10 000 个连接。这会造成服务器性能的严重下降。对高负荷的代理来说也同样如此。

实际上，浏览器确实使用了并行连接，但它们会将并行连接的总数限制为一个较小的值（通常是 4 个）。服务器可以随意关闭来自特定客户端的超量连接。

4.4.3　并行连接可能让人“感觉”更快一些

好了，这样看来并行连接并不总是能使页面加载得更快一些。但如前所述，即使实际上它们并没有加快页面的传输速度，并行连接通常也会让用户觉得页面加载得更

注 14：实际上，多条连接会产生一些额外的开销，使用并行连接装载整个页面所需的时间很可能比串行下载的时间更长。

快了，因为多个组件对象同时出现在屏幕上时，用户能够看到加载的进展。[15] 如果整个屏幕上有很多动作在进行，即使实际上秒表显示整个页面的下载时间更长，人们也会认为 Web 页面加载得更快一些。

4.5 持久连接

Web 客户端经常会打开到同一个站点的连接。比如，一个 Web 页面上的大部分内嵌图片通常都来自同一个 Web 站点，而且相当一部分指向其他对象的超链通常都指向同一个站点。因此，初始化了对某服务器 HTTP 请求的应用程序很可能会在不久的将来对那台服务器发起更多的请求（比如，获取在线图片）。这种性质被称为*站点局部性*（site locality）。

因此，HTTP/1.1（以及 HTTP/1.0 的各种增强版本）允许 HTTP 设备在事务处理结束之后将 TCP 连接保持在打开状态，以便为未来的 HTTP 请求重用现存的连接。在事务处理结束之后仍然保持在打开状态的 TCP 连接被称为*持久连接*。非持久连接会在每个事务结束之后关闭。持久连接会在不同事务之间保持打开状态，直到客户端或服务器决定将其关闭为止。

90

重用已对目标服务器打开的空闲持久连接，就可以避开缓慢的连接建立阶段。而且，已经打开的连接还可以避免慢启动的拥塞适应阶段，以便更快速地进行数据的传输。

4.5.1 持久以及并行连接

我们看到，并行连接可以提高复合页面的传输速度。但并行连接也有一些缺点。

- 每个事务都会打开 / 关闭一条新的连接，会耗费时间和带宽。
- 由于 TCP 慢启动特性的存在，每条新连接的性能都会有所降低。
- 可打开的并行连接数量实际上是有限的。

持久连接有一些比并行连接更好的地方。持久连接降低了时延和连接建立的开销，将连接保持在已调谐状态，而且减少了打开连接的潜在数量。但是，管理持久连接时要特别小心，不然就会累积出大量的空闲连接，耗费本地以及远程客户端和服务器上的资源。

持久连接与并行连接配合使用可能是最高效的方式。现在，很多 Web 应用程序都会打开少量的并行连接，其中的每一个都是持久连接。持久连接有两种类型：比较老

注 15：渐进式图片会先显示低分辨率的近似图形，然后再逐渐增加图片的分辨率，而随着渐进式图片应用的逐步增加，这种效果就更加明显了。

的 HTTP/1.0+“keep-alive”连接，以及现代的 HTTP/1.1“persistent”连接。在接下来的几节中我们将对这两种类型进行介绍。

4.5.2　HTTP/1.0+ keep-alive连接

大约从 1996 年开始，很多 HTTP/1.0 浏览器和服务器都进行了扩展，以支持一种被称为 keep-alive 连接的早期实验型持久连接。这些早期的持久连接受到了一些互操作性设计方面问题的困扰，这些问题在后期的 HTTP/1.1 版本中都得到了修正，但很多客户端和服务器仍然在使用这些早期的 keep-alive 连接。

图 4-13 显示了 keep-alive 连接的一些性能优点，图中将在串行连接上实现 4 个 HTTP 事务的时间线与在一条持久连接上实现同样事务所需的时间线进行了比较。由于去除了进行连接和关闭连接的开销，所以时间线有所缩减。[16] 91

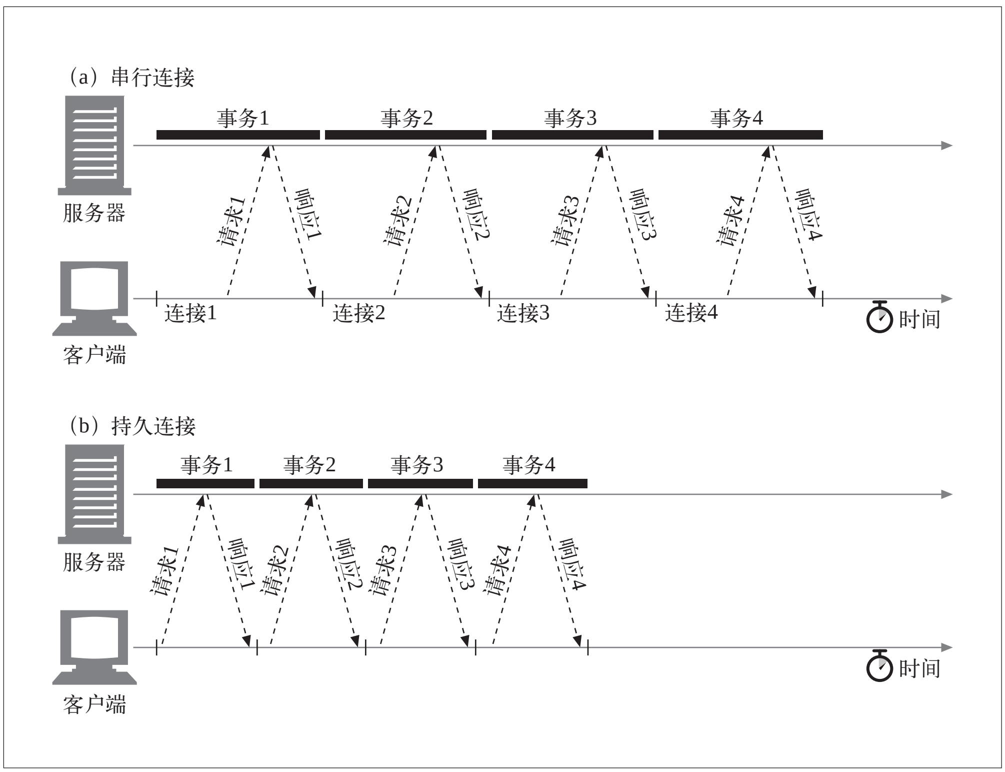


图 4-13　4 个事务（串行与持久连接）

注 16：由于去除了慢启动阶段，请求和响应时间可能也有缩减。这种性能收益在图中没有显示出来。

4.5.3 Keep-Alive操作

keep-alive 已经不再使用了，而且在当前的 HTTP/1.1 规范中也没有对它的说明了。但浏览器和服务器对 keep-alive 握手的使用仍然相当广泛，因此，HTTP 的实现者应该做好与之进行交互操作的准备。现在我们来快速浏览一下 keep-alive 的操作。对 keep-alive 握手更详细的解释请参见较早的 HTTP/1.1 规范版本（比如 RFC 2068）。

实现 HTTP/1.0 keep-alive 连接的客户端可以通过包含 `Connection: Keep-Alive` 首部请求将一条连接保持在打开状态。

如果服务器愿意为下一条请求将连接保持在打开状态，就在响应中包含相同的首部（参见图 4-14）。如果响应中没有 `Connection: Keep-Alive` 首部，客户端就认为服务器不支持 keep-alive，会在发回响应报文之后关闭连接。

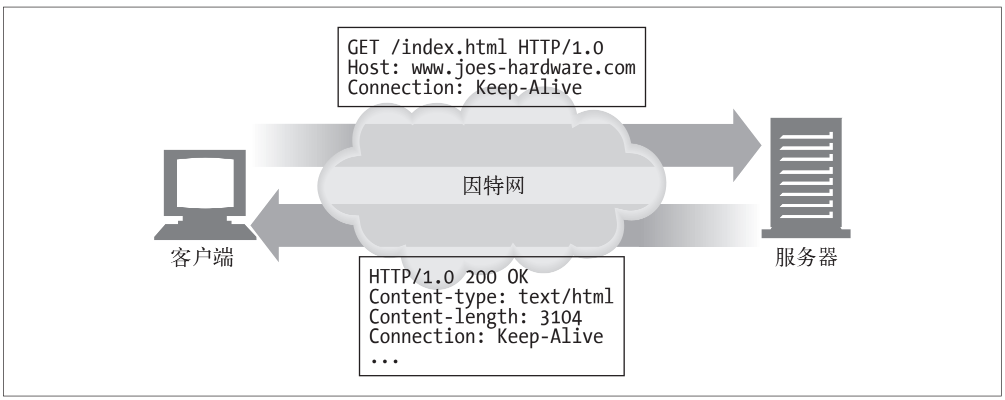


图 4-14　HTTP/1.0 keep-alive 事务首部的握手过程

4.5.4 Keep-Alive选项

注意，`keep-Alive` 首部只是请求将连接保持在活跃状态。发出 keep-alive 请求之
92 后，客户端和服务器并不一定会同意进行 keep-alive 会话。它们可以在任意时刻关闭空闲的 keep-alive 连接，并可随意限制 keep-alive 连接所处理事务的数量。

可以用 `Keep-Alive` 通用首部中指定的、由逗号分隔的选项来调节 keep-alive 的行为。

- 参数 `timeout` 是在 `Keep-Alive` 响应首部发送的。它估计了服务器希望将连接保持在活跃状态的时间。这并不是一个承诺值。
- 参数 `max` 是在 `Keep-Alive` 响应首部发送的。它估计了服务器还希望为多少个事务保持此连接的活跃状态。这并不是一个承诺值。

- Keep-Alive 首部还可支持任意未经处理的属性，这些属性主要用于诊断和调试。语法为 name [=value]。

Keep-Alive 首部完全是可选的，但只有在提供 Connection: Keep-Alive 时才能使用它。这里有个 Keep-Alive 响应首部的例子，这个例子说明服务器最多还会为另外 5 个事务保持连接的打开状态，或者将打开状态保持到连接空闲了 2 分钟之后。

```
Connection: Keep-Alive
Keep-Alive: max=5, timeout=120
```

4.5.5 Keep-Alive连接的限制和规则

使用 keep-alive 连接时有一些限制和一些需要澄清的地方。

- 在 HTTP/1.0 中，keep-alive 并不是默认使用的。客户端必须发送一个 Connection: Keep-Alive 请求首部来激活 keep-alive 连接。
- Connection: Keep-Alive 首部必须随所有希望保持持久连接的报文一起发送。如果客户端没有发送 Connection: Keep-Alive 首部，服务器就会在那条请求之后关闭连接。 93
- 通过检测响应中是否包含 Connection: Keep-Alive 响应首部，客户端可以判断服务器是否会在发出响应之后关闭连接。
- 只有在无需检测到连接的关闭即可确定报文实体主体部分长度的情况下，才能将连接保持在打开状态——也就是说实体的主体部分必须有正确的 Content-Length，有多部件媒体类型，或者用分块传输编码的方式进行了编码。在一条 keep-alive 信道中回送错误的 Content-Length 是很糟糕的事，这样的话，事务处理的另一端就无法精确地检测出一条报文的结束和另一条报文的开始了。
- 代理和网关必须执行 Connection 首部的规则。代理或网关必须在将报文转发出去或将其高速缓存之前，删除在 Connection 首部中命名的所有首部字段以及 Connection 首部自身。
- 严格来说，不应该与无法确定是否支持 Connection 首部的代理服务器建立 keep-alive 连接，以防止出现下面要介绍的哑代理问题。在实际应用中不是总能做到这一点的。
- 从技术上来讲，应该忽略所有来自 HTTP/1.0 设备的 Connection 首部字段（包括 Connection: Keep-Alive），因为它们可能是由比较老的代理服务器误转发的。但实际上，尽管可能会有在老代理上挂起的危险，有些客户端和服务器还是会违反这条规则。
- 除非重复发送请求会产生其他一些副作用，否则如果在客户端收到完整的响应之前连接就关闭了，客户端就一定要做好重试请求的准备。

4.5.6 Keep-Alive和哑代理

我们来仔细看看 keep-alive 和哑代理中一些比较微妙的问题。Web 客户端的 Connection: Keep-Alive 首部应该只会对这条离开客户端的 TCP 链路产生影响。这就是将其称作“连接”首部的原因。如果客户端正在与一台 Web 服务器对话，客户端可以发送一个 Connection: Keep-Alive 首部来告知服务器它希望保持连接的活跃状态。如果服务器支持 keep-alive，就回送一个 Connection: Keep-Alive 首部，否则就不回送。

1. Connection首部和盲中继

问题出在代理上——尤其是那些不理解 Connection 首部，而且不知道在沿着转发链路将其发送出去之前，应该将该首部删除的代理。很多老的或简单的代理都是盲中继（blind relay），它们只是将字节从一个连接转发到另一个连接中去，不对 Connection 首部进行特殊的处理。

假设有一个 Web 客户端正通过一个作为盲中继使用的哑代理与 Web 服务器进行对话。图 4-15 显示的就是这种情形。

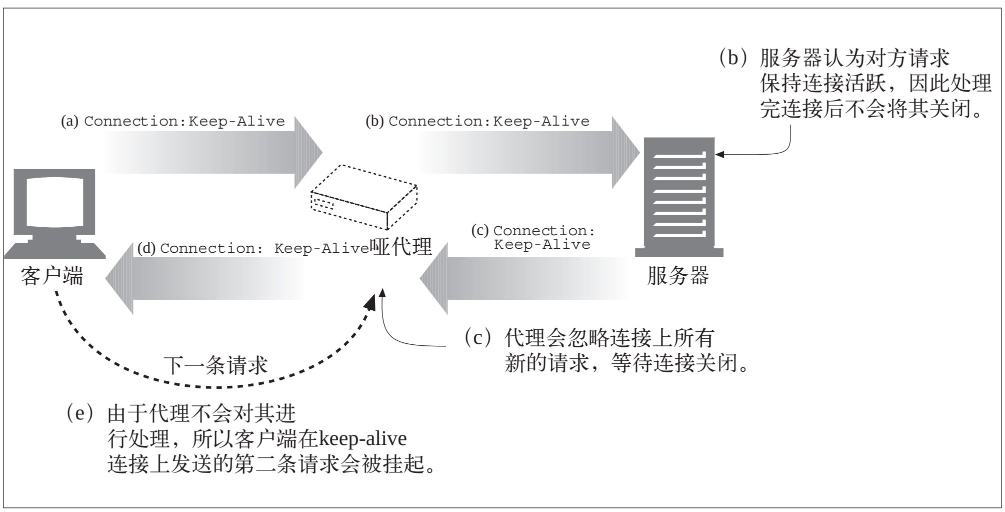


图 4-15　keep-alive 无法与不支持 Connection 首部的代理进行互操作

这幅图中发生的情况如下所示。

(1) 在图 4-15a 中，Web 客户端向代理发送了一条报文，其中包含了 Connection: Keep-Alive 首部，如果可能的话请求建立一条 keep-alive 连接。客户端等待响应，以确定对方是否认可它对 keep-alive 信道的请求。

94

(2) 哑代理收到了这条 HTTP 请求，但它并不理解 Connection 首部（只是将其作为一个扩展首部对待）。代理不知道 keep-alive 是什么意思，因此只是沿着转发链路将报文一字不漏地发送给服务器（图 4-15b）。但 Connection 首部是个逐跳首部，只适用于单条传输链路，不应该沿着传输链路向下传输。接下来，就要发生一些很糟糕的事情了。

(3) 在图 4-15b 中，经过中继的 HTTP 请求抵达了 Web 服务器。当 Web 服务器收到经过代理转发的 Connection: Keep-Alive 首部时，会误以为代理（对服务器来说，这个代理看起来就和所有其他客户端一样）希望进行 keep-alive 对话！对 Web 服务器来说这没什么问题——它同意进行 keep-alive 对话，并在图 4-15c 中回送了一个 Connection: Keep-Alive 响应首部。所以，此时 Web 服务器认为它在与代理进行 keep-alive 对话，会遵循 keep-alive 的规则。但代理却对 keep-alive 一无所知。不妙。

(4) 在图 4-15d 中，哑代理将 Web 服务器的响应报文回送给客户端，并将来自 Web 服务器的 Connection: Keep-Alive 首部一起传送过去。客户端看到这个首部，就会认为代理同意进行 keep-alive 对话。所以，此时客户端和服务器都认为它们在进行 keep-alive 对话，但与它们进行对话的代理却对 keep-alive 一无所知。

(5) 由于代理对 keep-alive 一无所知，所以会将收到的所有数据都回送给客户端，然后等待源端服务器关闭连接。但源端服务器会认为代理已经显式地请求它将连接保持在打开状态了，所以不会去关闭连接。这样，代理就会挂在那里等待连接的关闭。

(6) 客户端在图 4-15d 中收到了回送的响应报文时，会立即转向下一条请求，在 keep-alive 连接上向代理发送另一条请求（参见图 4-15e）。而代理并不认为同一条连接上会有其他请求到来，请求被忽略，浏览器就在这里转圈，不会有任何进展了。 95

(7) 这种错误的通信方式会使浏览器一直处于挂起状态，直到客户端或服务器将连接超时，并将其关闭为止。[17]

2. 代理和逐跳首部

为避免此类代理通信问题的发生，现代的代理都绝不能转发 Connection 首部和所有名字出现在 Connection 值中的首部。因此，如果一个代理收到了一个 Connection: Keep-Alive 首部，是不应该转发 Connection 首部，或所有名为 Keep-Alive 的首部的。

注 17：在很多类似的情形下，盲中继和转发的握手信息都会引发问题。

另外，还有几个不能作为 Connection 首部值列出，也不能被代理转发或作为缓存响应使用的首部。其中包括 Proxy-Authenticate、Proxy-Connection、Transfer-Encoding 和 Upgrade。更多信息，请参考 4.3.1 节。

4.5.7 插入Proxy-Connection

Netscape 的浏览器及代理实现者们提出了一个对盲中继问题的变通做法，这种做法并不要求所有的 Web 应用程序支持高版本的 HTTP。这种变通做法引入了一个名为 Proxy-Connection 的新首部，解决了在客户端后面紧跟着一个盲中继所带来的问题——但并没有解决所有其他情况下存在的问题。在显式配置了代理的情况下，现代浏览器都实现了 Proxy-Connection，很多代理都能够理解它。

问题是哑代理盲目地转发 Connection: Keep-Alive 之类的逐跳首部惹出了麻烦。逐跳首部只与一条特定的连接有关，不能被转发。当下游服务器误将转发来的首部作为来自代理自身的请求解释，用它来控制自己的连接时，就会引发问题。

在网景的变通做法是，浏览器会向代理发送非标准的 Proxy-Connection 扩展首部，而不是官方支持的著名的 Connection 首部。如果代理是盲中继，它会将无意义的 Proxy-Connection 首部转发给 Web 服务器，服务器会忽略此首部，不会带来任何问题。但如果代理是个聪明的代理（能够理解持久连接的握手动作），就用一个 Connection 首部取代无意义的 Proxy-Connection 首部，然后将其发送给服务器，以收到预期的效果。

图 4-16a ～图 4-16d 显示了盲中继是如何向 Web 服务器转发 Proxy-Connection
96 首部，而不带来任何问题的，Web 服务器忽略了这个首部，这样在客户端和代理，或者代理和服务器之间就不会建立起 keep-alive 连接了。图 4-16e ～图 4-16h 中那个聪明的代理知道 Proxy-Connection 首部是对 keep-alive 对话的请求，它会发送自己的 Connection: Keep-Alive 首部来建立 keep-alive 连接。

在客户端和服务器之间只有一个代理时可以用这种方案来解决问题。但如图 4-17 所示，如果在哑代理的任意一侧还有一个聪明的代理，这个问题就会再次露头了。

而且，网络中出现“不可见”代理的情况现在变得很常见了，这些代理可以是防火墙、拦截缓存，或者是反向代理服务器的加速器。这些设备对浏览器是不可见的，所以浏览器不会向它们发送 Proxy-Connection 首部。透明的 Web 应用程序正确地实现持久连接是非常重要的。

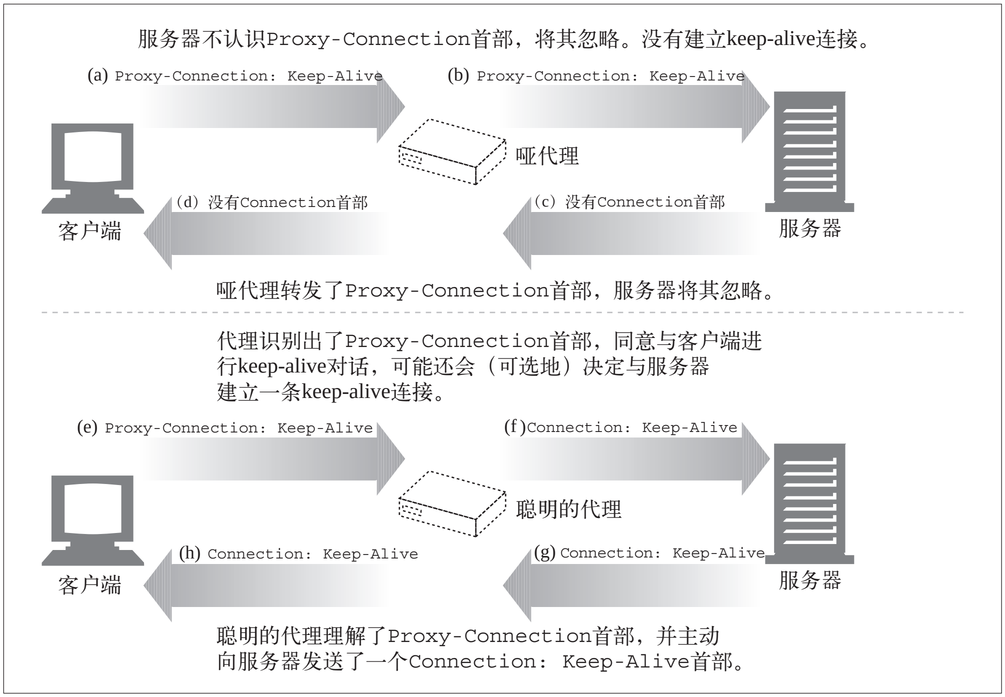


图 4-16　**Proxy-Connection** 首部修正了单个盲中继带来的问题

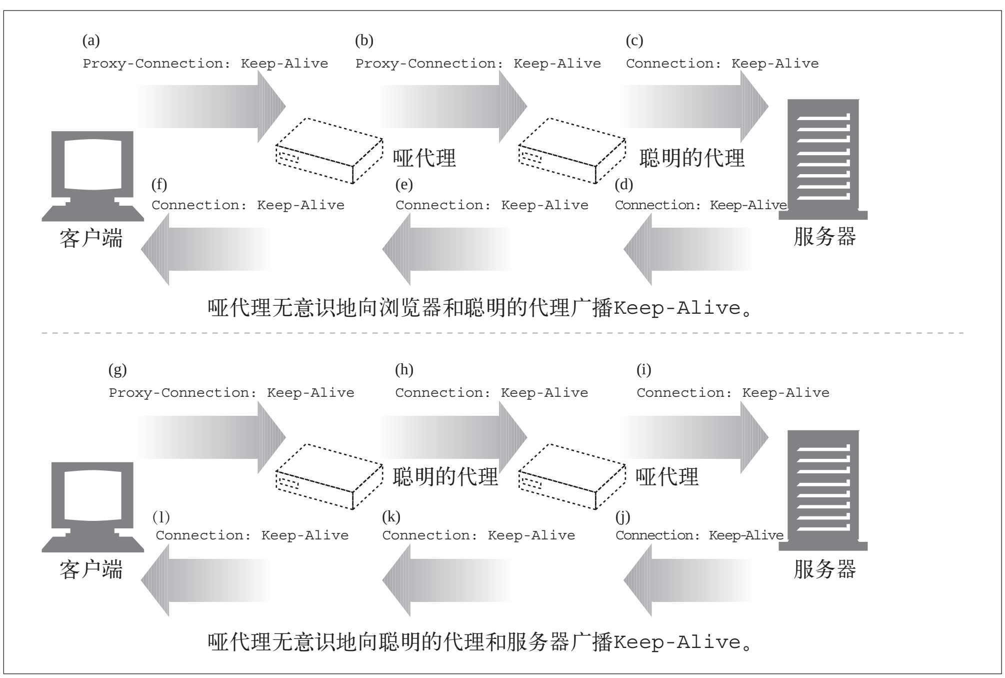


图 4-17　对有多层次代理的情况，**Proxy-Connection** 仍然无法解决问题

4.5.8 HTTP/1.1持久连接

HTTP/1.1 逐渐停止了对 keep-alive 连接的支持，用一种名为持久连接（persistent connection）的改进型设计取代了它。持久连接的目的与 keep-alive 连接的目的相
97 同，但工作机制更优一些。

与 HTTP/1.0+ 的 keep-alive 连接不同，HTTP/1.1 持久连接在默认情况下是激活的。除非特别指明，否则 HTTP/1.1 假定所有连接都是持久的。要在事务处理结束之后将连接关闭，HTTP/1.1 应用程序必须向报文中显式地添加一个 `Connection: close` 首部。这是与以前的 HTTP 协议版本很重要的区别，在以前的版本中，keep-alive 连接要么是可选的，要么根本就不支持。

HTTP/1.1 客户端假定在收到响应后，除非响应中包含了 `Connection: close` 首部，不然 HTTP/1.1 连接就仍维持在打开状态。但是，客户端和服务器仍然可以随时关闭空闲的连接。不发送 `Connection: close` 并不意味着服务器承诺永远将连接保持在打开状态。

4.5.9 持久连接的限制和规则

在持久连接的使用中有以下限制和需要澄清的问题。

- 发送了 `Connection: close` 请求首部之后，客户端就无法在那条连接上发送更多的请求了。
- 如果客户端不想在连接上发送其他请求了，就应该在最后一条请求中发送一个 `Connection: close` 请求首部。
- 只有当连接上所有的报文都有正确的、自定义报文长度时——也就是说，实体主体部分的长度都和相应的 `Content-Length` 一致，或者是用分块传输编码方式
98 编码的——连接才能持久保持。
- HTTP/1.1 的代理必须能够分别管理与客户端和服务器的持久连接——每个持久连接都只适用于一跳传输。
- （由于较老的代理会转发 `Connection` 首部，所以）HTTP/1.1 的代理服务器不应该与 HTTP/1.0 客户端建立持久连接，除非它们了解客户端的处理能力。实际上，这一点是很难做到的，很多厂商都违背了这一原则。
- 尽管服务器不应该试图在传输报文的过程中关闭连接，而且在关闭连接之前至少应该响应一条请求，但不管 `Connection` 首部取了什么值，HTTP/1.1 设备都可以在任意时刻关闭连接。

- HTTP/1.1 应用程序必须能够从异步的关闭中恢复出来。只要不存在可能会累积起来的副作用，客户端都应该重试这条请求。
- 除非重复发起请求会产生副作用，否则如果在客户端收到整条响应之前连接关闭了，客户端就必须要重新发起请求。
- 一个用户客户端对任何服务器或代理最多只能维护两条持久连接，以防服务器过载。代理可能需要更多到服务器的连接来支持并发用户的通信，所以，如果有 N 个用户试图访问服务器的话，代理最多要维持 $2N$ 条到任意服务器或父代理的连接。

4.6 管道化连接

HTTP/1.1 允许在持久连接上可选地使用*请求管道*。这是相对于 keep-alive 连接的又一性能优化。在响应到达之前，可以将多条请求放入队列。当第一条请求通过网络流向地球另一端的服务器时，第二条和第三条请求也可以开始发送了。在高时延网络条件下，这样做可以降低网络的环回时间，提高性能。

图 4-18a-c 显示了持久连接是怎样消除 TCP 连接时延，以及管道化请求（参见图 4-18c）是如何消除传输时延的。

对管道化连接有几条限制。

- 如果 HTTP 客户端无法确认连接是持久的，就不应该使用管道。
- 必须按照与请求相同的顺序回送 HTTP 响应。HTTP 报文中没有序列号标签，因此如果收到的响应失序了，就没办法将其与请求匹配起来了。
- HTTP 客户端必须做好连接会在任意时刻关闭的准备，还要准备好重发所有未完成的管道化请求。如果客户端打开了一条持久连接，并立即发出了 10 条请求， 99
  服务器可能在只处理了，比方说，5 条请求之后关闭连接。剩下的 5 条请求会失败，客户端必须能够应对这些过早关闭连接的情况，重新发出这些请求。
- HTTP 客户端不应该用管道化的方式发送会产生副作用的请求（比如 POST）。总之，出错的时候，管道化方式会阻碍客户端了解服务器执行的是一系列管道化请求中的哪一些。由于无法安全地重试 POST 这样的非幂等请求，所以出错时，就存在某些方法永远不会被执行的风险。 100

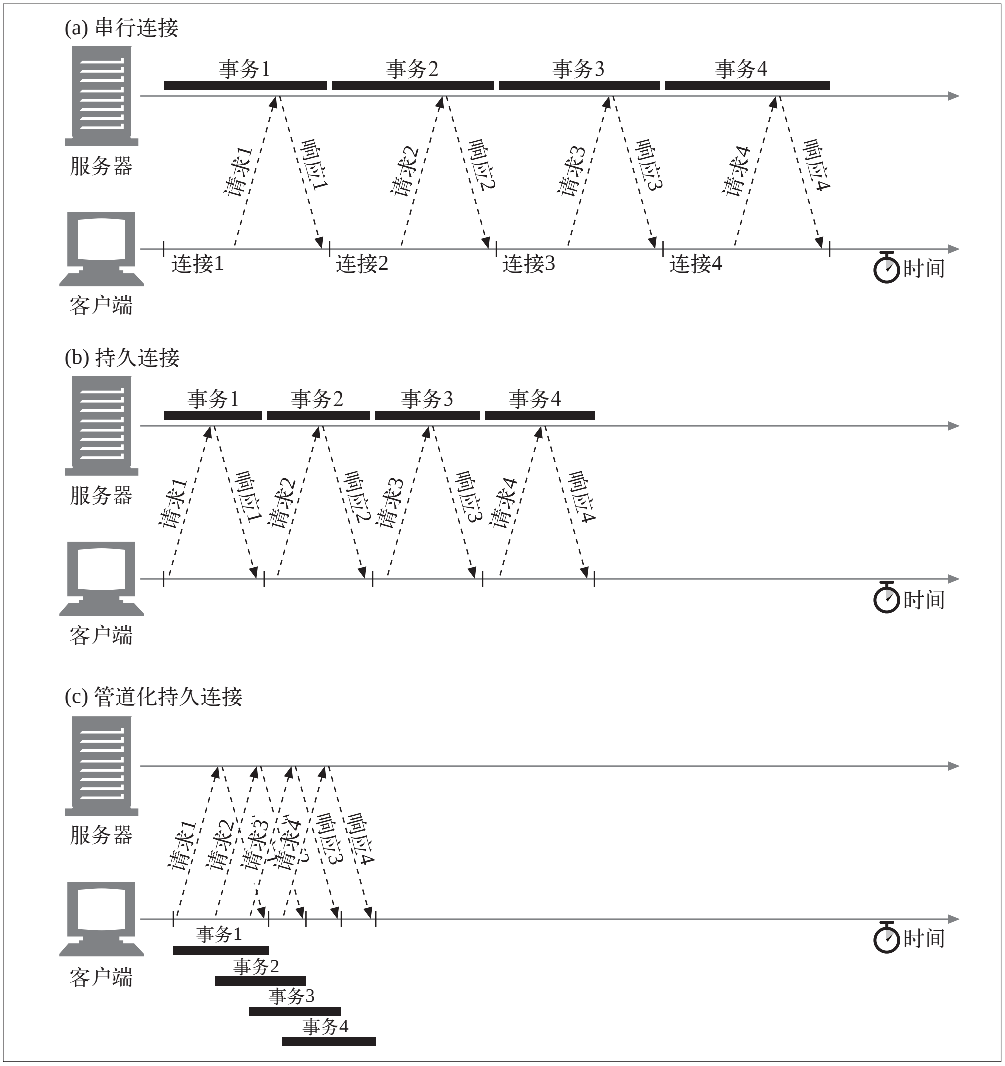


图 4-18　4 个事务（管道化连接）

4.7　关闭连接的奥秘

连接管理——尤其是知道在什么时候以及如何去关闭连接——是 HTTP 的实用魔法之一。这个问题比很多开发者起初意识到的复杂一些，而且没有多少资料涉及这个问题。

4.7.1　“任意”解除连接

所有 HTTP 客户端、服务器或代理都可以在任意时刻关闭一条 TCP 传输连接。通常会在一条报文结束时关闭连接，[18] 但出错的时候，也可能在首部行的中间，或其他

奇怪的地方关闭连接。

对管道化持久连接来说，这种情形是很常见的。HTTP 应用程序可以在经过任意一段时间之后，关闭持久连接。比如，在持久连接空闲一段时间之后，服务器可能会决定将其关闭。

但是，服务器永远都无法确定在它关闭"空闲"连接的那一刻，在线路那一头的客户端有没有数据要发送。如果出现这种情况，客户端就会在写入半截请求报文时发现出现了连接错误。

4.7.2 Content-Length及截尾操作

每条 HTTP 响应都应该有精确的 `Content-Length` 首部，用以描述响应主体的尺寸。一些老的 HTTP 服务器会省略 `Content-Length` 首部，或者包含错误的长度指示，这样就要依赖服务器发出的连接关闭来说明数据的真实末尾。

客户端或代理收到一条随连接关闭而结束的 HTTP 响应，且实际传输的实体长度与 `Content-Length` 并不匹配（或没有 `Content-Length`）时，接收端就应该质疑长度的正确性。

如果接收端是个缓存代理，接收端就不应该缓存这条响应（以降低今后将潜在的错误报文混合起来的可能）。代理应该将有问题的报文原封不动地转发出去，而不应该试图去"校正"`Content-Length`，以维护语义的透明性。

4.7.3 连接关闭容限、重试以及幂等性

即使在非错误情况下，连接也可以在任意时刻关闭。HTTP 应用程序要做好正确处理非预期关闭的准备。如果在客户端执行事务的过程中，传输连接关闭了，那么，除非事务处理会带来一些副作用，否则客户端就应该重新打开连接，并重试一次。对管道化连接来说，这种情况更加严重一些。客户端可以将大量请求放入队列中排 101
队，但源端服务器可以关闭连接，这样就会留下大量未处理的请求，需要重新调度。

副作用是很重要的问题。如果在发送出一些请求数据之后，收到返回结果之前，连接关闭了，客户端就无法百分之百地确定服务器端实际激活了多少事务。有些事务，比如 GET 一个静态的 HTML 页面，可以反复执行多次，也不会有什么变化。而其他一些事务，比如向一个在线书店 POST 一张订单，就不能重复执行，不然会有下多张订单的危险。

注 18：除非服务器怀疑出现了客户端或网络故障，否则就不应该在请求的中间关闭连接。

如果一个事务，不管是执行一次还是很多次，得到的结果都相同，这个事务就是幂等的。实现者们可以认为 GET、HEAD、PUT、DELETE、TRACE 和 OPTIONS 方法都共享这一特性。[19] 客户端不应该以管道化方式传送非幂等请求（比如 POST）。否则，传输连接的过早终止就会造成一些不确定的后果。要发送一条非幂等请求，就需要等待来自前一条请求的响应状态。

尽管用户 Agent 代理可能会让操作员来选择是否对请求进行重试，但一定不能自动重试非幂等方法或序列。比如，大多数浏览器都会在重载一个缓存的 POST 响应时提供一个对话框，询问用户是否希望再次发起事务处理。

4.7.4 正常关闭连接

如图 4-19 所示，TCP 连接是双向的。TCP 连接的每一端都有一个输入队列和一个输出队列，用于数据的读或写。放入一端输出队列中的数据最终会出现在另一端的输入队列中。

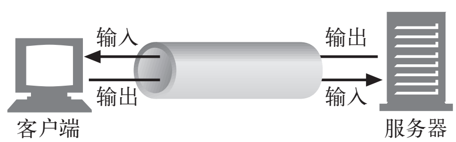


图 4-19 TCP 连接是双向的

1. 完全关闭与半关闭

应用程序可以关闭 TCP 输入和输出信道中的任意一个，或者将两者都关闭了。套
102 接字调用 `close()` 会将 TCP 连接的输入和输出信道都关闭了。这被称作“完全关闭”，如图 4-20a 所示。还可以用套接字调用 `shutdown()` 单独关闭输入或输出信道。这被称为“半关闭”，如图 4-20b 所示。

2. TCP关闭及重置错误

简单的 HTTP 应用程序可以只使用完全关闭。但当应用程序开始与很多其他类型的 HTTP 客户端、服务器和代理进行对话且开始使用管道化持久连接时，使用半关闭来防止对等实体收到非预期的写入错误就变得很重要了。

注 19：基于 GET 构建动态表单的管理者们要确保这些表单是幂等的。

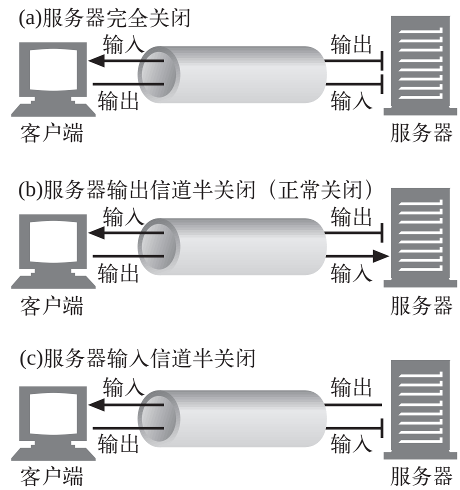


图 4-20　完全关闭和半关闭

总之，关闭连接的输出信道总是很安全的。连接另一端的对等实体会在从其缓冲区中读出所有数据之后收到一条通知，说明流结束了，这样它就知道你将连接关闭了。

关闭连接的输入信道比较危险，除非你知道另一端不打算再发送其他数据了。如果另一端向你已关闭的输入信道发送数据，操作系统就会向另一端的机器回送一条 TCP“连接被对端重置”的报文，如图 4-21 所示。大部分操作系统都会将这种情况作为很严重的错误来处理，删除对端还未读取的所有缓存数据。对管道化连接来说，这是非常糟糕的事情。

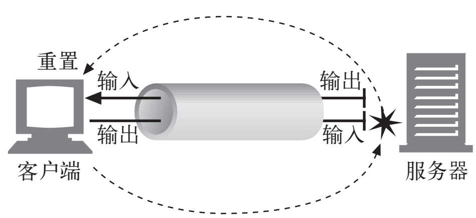


图 4-21　将数据传送到已关闭连接时会产生“连接被对端重置”错误

比如你已经在一条持久连接上发送了 10 条管道式请求了，响应也已经收到了，正在操作系统的缓冲区中存着呢（但应用程序还未将其读走）。现在，假设你发送了第 11 条请求，但服务器认为你使用这条连接的时间已经够长了，决定将其关闭。那么你的第 11 条请求就会被发送到一条已关闭的连接上去，并会向你回送一条重置信息。这个重置信息会清空你的输入缓冲区。 103

当你最终要去读取数据的时候，会得到一个连接被对端重置的错误，已缓存的未读响应数据都丢失了，尽管其中的大部分都已经成功抵达你的机器了。

3. 正常关闭

HTTP 规范建议，当客户端或服务器突然要关闭一条连接时，应该“正常地关闭传输连接”，但它并没有说明应该如何去做。

总之，实现正常关闭的应用程序首先应该关闭它们的输出信道，然后等待连接另一端的对等实体关闭它的输出信道。当两端都告诉对方它们不会再发送任何数据（比如关闭输出信道）之后，连接就会被完全关闭，而不会有重置的危险。

但不幸的是，无法确保对等实体会实现半关闭，或对其进行检查。因此，想要正常关闭连接的应用程序应该先半关闭其输出信道，然后周期性地检查其输入信道的状态（查找数据，或流的末尾）。如果在一定的时间区间内对端没有关闭输入信道，应用程序可以强制关闭连接，以节省资源。

4.8 更多信息

到这里我们对 HTTP 通道事务处理的介绍就结束了。更多有关 TCP 性能和 HTTP 连接管理功能的内容请参见下列参考资源。

4.8.1 HTTP连接

- http://www.ietf.org/rfc/rfc2616.txt
  RFC 2616，“超文本传输协议——HTTP/1.1”是 HTTP/1.1 的官方规范；解释了
  104 并行、持久和管道式 HTTP 连接的使用，以及用于实现这些连接的 HTTP 首部字段。此文档并未涵盖对底层 TCP 连接的正确使用。

- http://www.ietf.org/rfc/rfc2068.txt
  RFC 2068 是 HTTP/1.1 协议的 1997 年的版本。其中包含了 RFC 2616 中没有的、对 HTTP/1.0+ keep-alive 连接的解释。

- http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-connection-00.txt
  这个过期的因特网草案“HTTP Connection Management”,（“HTTP 连接管理”）探讨了 HTTP 连接管理面临的问题。

4.8.2 HTTP性能问题

- http://www.w3.org/Protocols/HTTP/Performance/

这个名为“HTTP Performance Overview”（“HTTP 性能概览”）的 W3C Web 页面包含了几篇与 HTTP 性能和连接管理有关的文章和一些工具。

- http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html
  这个由 Simon Spero 制作的简短备忘录“Analysis of HTTP Performance Problems”(“HTTP 性能问题分析”) 是最早（1994 年）对 HTTP 连接性能进行评估的文献之一。对早期由于缺乏连接建立、慢启动和连接共享所造成的影响进行了一些性能测试，这个备忘录给出了一些测试结果。

- ftp://gatekeeper.dec.com/pub/DEC/WRL/research-reports/WRL-TR-95.4.pdf
  “The Case for Persistent-Connection HTTP.”（“持久连接 HTTP 的实例。”）

- http://www.isi.edu/lsam/publications/phttp_tcp_interactions/paper.html
  “Performance Interactions Between P-HTTP and TCP” Implementations（“P-HTTP 和 TCP 实现之间的性能交互”）。

- http://www.sun.com/sun-on-net/performance/tcp.slowstart.html
  “TCP Slow Start Tuning for Solaris”（“Solaris 的 TCP 慢启动调试”）是 Sun 微系统公司的一个 Web 页面，介绍了 TCP 慢启动带来的一些实际的影响。即使工作在不同的操作系统上，读一读这份资料也是有所帮助的。

4.8.3 TCP/IP

下面三本 W. Richard Stevens 的书都非常棒，详细介绍了 TCP/IP 的工程问题。对使用 TCP 的人来说尤其有用：

- *TCP Illustrated, Volume 1: The Protocols*[20]（《TCP 详解，卷 1：协议》）
  W. Richard Stevens，Addison Wesley 公司出版。

- *UNIX Network Programming, Volume 1: Networking APIs*[21]（《UNIX 网络编程，卷 1：套接字联网 API（第 3 版）》）
  W. Richard Stevens，Prentice-Hall 公司出版。

- *UNIX Network Programming, Volume 2: The Implementation*[22]（《UNIX 网络编程，卷 2：进程间通信（第 2 版）》）
  W. Richard Stevens，Prentice-Hall 公司出版。 105

注 20：本书影印版已由人民邮电出版社出版。（编者注）
注 21~22：两本书中文版已由人民邮电出版社出版。（编者注）

下面的文章和规范介绍了 TCP/IP 及影响其性能的特性。其中有些规范已经有 20 多年的历史了，鉴于 TCP/IP 在全球范围内的成功，很可能已经可以将其归为历史宝藏了。

- http://www.acm.org/sigcomm/ccr/archive/2001/jan01/ccr-200101-mogul.pdf
  在“Rethinking the TCP Nagle Algorithm”（“对 TCP Nagle 算法的反思”）一文中，Jeff Mogul 和 Greg Minshall 提出了 Nagle 算法的一种现代视角，概括了哪些应用程序应该，哪些不应该使用这个算法，并提出了几条改进意见。

- http://www.ietf.org/rfc/rfc2001.txt
  RFC 2001，“TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”（“TCP 慢启动、避免拥塞、快速重传以及快速恢复算法”）定义了 TCP 慢启动算法。

- http://www.ietf.org/rfc/rfc1122.txt
  RFC 1122，“Requirements for Internet Hosts—Communication Layers”（“对因特网主机的要求——通信层”）讨论了 TCP 确认和延迟确认。

- http://www.ietf.org/rfc/rfc896.txt
  RFC 896，“Congestion Control in IP/TCP Internetworks”（“IP/TCP 网络间的拥塞控制”）是 John Nagle 于 1984 年发布的。描述了 TCP 拥塞控制的必要性。介绍了现在称为“Nagle 算法”的算法。

- http://www.ietf.org/rfc/rfc0813.txt
  RFC 813，“Window and Acknowledgement Strategy in TCP”（“TCP 中的窗口和确认机制”）是一个早些年（1982 年）制定的规范，它描述了 TCP 窗口和确认的实现机制，解释了延迟确认技术的早期技术。

- http://www.ietf.org/rfc/rfc0793.txt
  RFC 793，“Transmission Control Protocol”（“传输控制协议”），是 Jon Postel 于
  106 1981 年给出的 TCP 协议经典定义。

第二部分

HTTP结构

第二部分的 6 章主要介绍了 HTTP 服务器、代理、缓存、网关和机器人应用程序，这些都是 Web 系统架构的构造模块。

- 第 5 章概述了 Web 服务器结构。
- 第 6 章详细介绍了 HTTP 代理服务器，它们是连接 HTTP 客户端的中间服务器，是 HTTP 服务和控制的平台。
- 第 7 章深入研究了 Web 的缓存机制。缓存是通过对常用文档进行本地复制来提高性能、减少流量的设备。
- 第 8 章介绍了一些应用程序，通过这些程序，HTTP 就可以与使用不同协议（比如 SSL 加密协议）的软件进行互操作了。
- 第 9 章介绍了 Web 客户端，结束了 HTTP 架构之旅。 107~108
- 第 10 章涵盖了 HTTP 未来发展的一些主题，特别介绍了 HTTP-NG 技术。

第5章

Web服务器

Web 服务器每天会分发出数十亿的 Web 页面。这些页面可以告诉你天气情况，装载在线商店的购物车，还能帮你找到许久未联系的高中同学。Web 服务器是万维网的骨干。本章将介绍以下话题。

- 对多种使用不同类型软硬件的 Web 服务器进行调查。
- 介绍如何用 Perl 编写简单的诊断性 Web 服务器。
- 一步一步地解释 Web 服务器是如何处理 HTTP 事务的。

为了对问题进行具体的说明，例子中使用了 Apache Web 服务器及其配置选项。

5.1　各种形状和尺寸的Web服务器

Web 服务器会对 HTTP 请求进行处理并提供响应。术语“Web 服务器”可以用来表示 Web 服务器的软件，也可以用来表示提供 Web 页面的特定设备或计算机。

Web 服务器有着不同的风格、形状和尺寸。有普通的 10 行 Perl 脚本的 Web 服务器、50MB 的安全商用引擎以及极小的卡上服务器。但不管功能有何差异，所有的 Web 服务器都能够接收请求资源的 HTTP 请求，将内容回送给客户端（参见图 1-5）。

5.1.1　Web服务器的实现

Web 服务器实现了 HTTP 和相关的 TCP 连接处理。负责管理 Web 服务器提供的资源，以及对 Web 服务器的配置、控制及扩展方面的管理。

Web 服务器逻辑实现了 HTTP 协议、管理着 Web 资源，并负责提供 Web 服务器的
109 管理功能。Web 服务器逻辑和操作系统共同负责管理 TCP 连接。底层操作系统负责管理底层计算机系统的硬件细节，并提供了 TCP/IP 网络支持、负责装载 Web 资源的文件系统以及控制当前计算活动的进程管理功能。

Web 服务器有各种不同的形式。

- 可以在标准的计算机系统上安装并运行通用的软件 Web 服务器。
- 如果不想那么麻烦地去安装软件，可以买一台 Web 服务器设备，通常会是一台安装在时髦机架上的计算机，里面的软件会预装并配置好。
- 随着微处理器奇迹般地出现，有些公司甚至可以在少量计算机芯片上实现嵌入式 Web 服务器，使其成为完美的（便携式）消费类设备管理控制台。

我们分别来看看这些实现方式。

5.1.2 通用软件Web服务器

通用软件 Web 服务器都运行在标准的、有网络功能的计算机系统上。可以选择开源软件（比如 Apache 或 W3C 的 Jigsaw）或者商业软件（比如微软和 iPlanet 的 Web 服务器）。基本上所有的计算机和操作系统中都有可用的 Web 服务器软件。

尽管不同类型的 Web 服务器程序有数万个（包括定制的和特殊用途的 Web 服务器），但大多数 Web 服务器软件都来自少数几个组织。

2002 年 2 月，Netcraft 调查（http://www.netcraft.com/survey/）显示有三家厂商主宰了公共因特网 Web 服务器市场（参见图 5-1）。

- 免费的 Apache 软件占据了所有因特网 Web 服务器中大约 60% 的市场。
- 微软的 Web 服务器占据了另外 30%。
- Sun 的 iPlanet 占据了另外 3%。

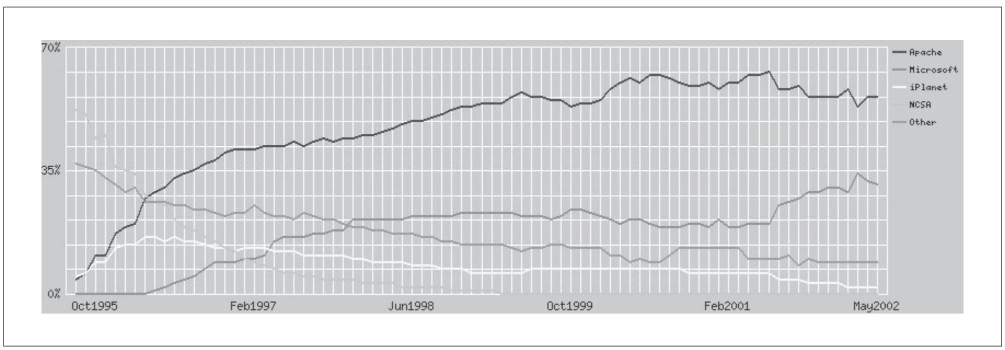


图 5-1　Netcraft 的自动化调查估计的 Web 服务器市场份额 110

但这些数据也不能尽信，通常大家都认为 Netcraft 调查会夸大 Apache 软件的优势。首先，在调查计算服务器的时候没有考虑其流行程度。各大 ISP 的代理服务器访问研究表明，Apache 服务器提供的页面数量远小于 60%，但仍然超过了微软和 Sun 的 iPlanet。然而，据说微软和 iPlanet 服务器在公司企业中要比 Apache 更受欢迎。

5.1.3 Web服务器设备

Web 服务器设备（Web server appliance）是预先打包好的软硬件解决方案。厂商会在他们选择的计算机平台上预先安装好软件服务器，并将软件配置好。下面是一些 Web 服务器设备的例子：

- Sun/Cobalt RaQ Web 设备（http://www.cobalt.com）；
- 东芝的 Magnia SG10（http://www.toshiba.com）；

- IBM 的 Whistle Web 服务器设备（http://www.whistle.com）。

应用解决方案不再需要安装及配置软件，通常可以极大地简化管理工作。但是，Web 服务器通常不太灵活，特性不太丰富，而且服务器硬件也不太容易重用或升级。

5.1.4　嵌入式Web服务器

嵌入式服务器（embeded server）是要嵌入到消费类产品（比如打印机或家用设备）中去的小型 Web 服务器。嵌入式 Web 服务器允许用户通过便捷的 Web 浏览器接口来管理其消费者设备。

有些嵌入式 Web 服务器甚至可以在小于一平方英寸的空间内实现，但通常只能提供最小特性功能集。下面是两种非常小的嵌入式 Web 服务器实例：

- IPic 火柴头大小的 Web 服务器（http://www-ccs.cs.umass.edu/~shri/iPic.html）；
- NetMedia SitePlayer SP1 以太网 Web 服务器（http://www.siteplayer.com）。

5.2　最小的Perl Web服务器

要构建一个特性完备的 HTTP 服务器，是需要做一些工作的。Apache Web 服务器的内核有超过 50 000 行的代码，那些可选处理模块的代码量更是远远超过这个数字。

这个软件所要做的就是支持 HTTP/1.1 的各种特性：丰富的资源支持、虚拟主机、访问控制、日志记录、配置、监视和性能特性。在这里，可以用少于 30 行的 Perl
111 代码来创建一个最小的可用 HTTP 服务器。我们来看看这是怎么实现的。

例 5-1 显示了一个名为 type-o-serve 的小型 Perl 程序。这个程序是个很有用的诊断工具，可以用来测试与客户端和代理的交互情况。与所有 Web 服务器一样，type-o-serve 会等待 HTTP 连接。只要 type-o-serve 收到了请求报文，就会将报文打印在屏幕上，然后等待用户输入（或粘贴）一条响应报文，并将其回送给客户端。通过这种方式，type-o-serve 假扮成一台 Web 服务器，记录下确切的 HTTP 请求报文，并允许用户回送任意的 HTTP 响应报文。

这个简单的 type-o-serve 实用程序并没有实现大部分的 HTTP 功能，但它是一种很有用的工具，产生服务器响应报文的方式与 Telnet 产生客户端请求报文的方式相同（参见例 5-1）。可以从 http://www.http-guide.com/tools/type-o-serve.pl 上下载 type-o-serve 程序。

例 5-1　type-o-serve——用于 HTTP 调试的最小型 Perl Web 服务器

```
#!/usr/bin/perl

use Socket;
use Carp;
use FileHandle;

# (1) use port 8080 by default, unless overridden on command line
$port = (@ARGV ? $ARGV[0] : 8080);

# (2) create local TCP socket and set it to listen for connections
$proto = getprotobyname('tcp');
socket(S, PF_INET, SOCK_STREAM, $proto) || die;
setsockopt(S, SOL_SOCKET, SO_REUSEADDR, pack("l", 1)) || die;
bind(S, sockaddr_in($port, INADDR_ANY)) || die;
listen(S, SOMAXCONN) || die;

# (3) print a startup message
printf("    <<<Type-O-Serve Accepting on Port %d>>>\n\n",$port);

while (1)
{
    # (4) wait for a connection C
    $cport_caddr = accept(C, S);
    ($cport,$caddr) = sockaddr_in($cport_caddr);
    C->autoflush(1);

    # (5) print who the connection is from
    $cname = gethostbyaddr($caddr,AF_INET);
    printf("    <<<Request From '%s'>>>\n",$cname);

    # (6) read request msg until blank line, and print on screen
    while ($line = <C>)
    {
        print $line;
        if ($line =~ /^\r/) { last; }
    }

    # (7) prompt for response message, and input response lines,
    #     sending response lines to client, until solitary "."
    printf("    <<<Type Response Followed by '.'>>>\n");

    while ($line = <STDIN>)
    {
        $line =~ s/\r//;
        $line =~ s/\n//;
        if ($line =~ /^\./) { last; }
        print C $line . "\r\n";
    }
    close(C);
}
```

112

图 5-2 显示了 Joe 的五金商店的管理员是如何用 type-o-serve 来测试 HTTP 通信的。

- 首先，管理员启动了 type-o-serve 诊断服务器，在一个特定的端口上监听。由于 Joe 的五金商店已经有一个产品化的 Web 服务器在监听 80 端口了，所以管理员用下面这条命令在端口 8080（可以选择任意未用端口）上启动了 type-o-serve 服务：

```
% type-o-serve.pl 8080
```

- 只要 type-o-serve 开始运行了，就可以将浏览器指向这个 Web 服务器。在图 5-2 中，浏览器指向了 http://www.joes-hardware.com:8080/foo/bar/blah.txt。
- type-o-serve 程序收到来自浏览器的 HTTP 请求报文，并将 HTTP 请求报文的内容打印在屏幕上。然后 type-o-serve 诊断工具会等待用户输入一条简单的响应报文，后面跟着只有一个句号的空行。
- type-o-serve 将 HTTP 响应报文回送给浏览器，浏览器会显示响应报文的主体。

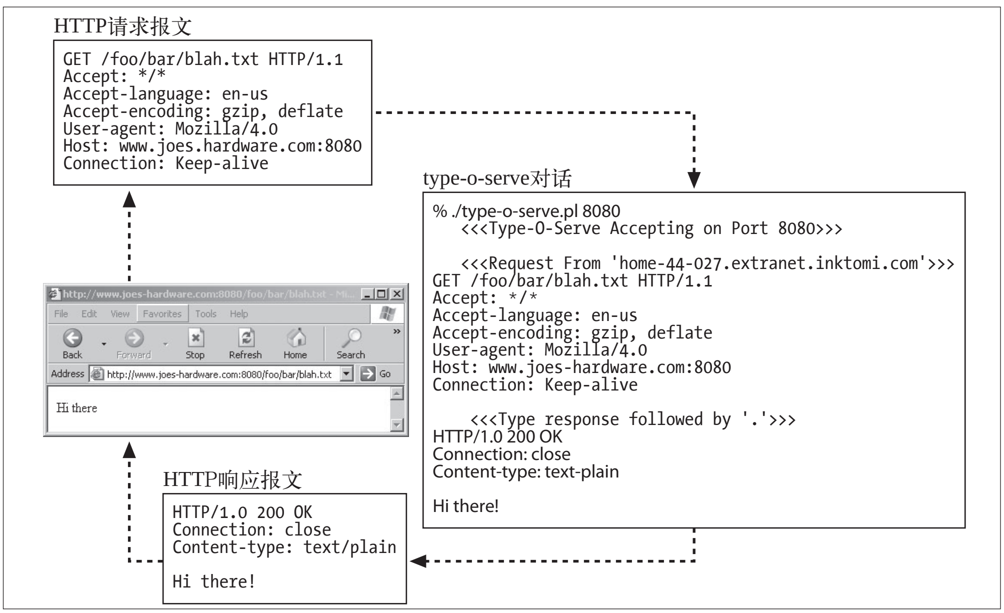


图 5-2　type-o-serve 实用程序让用户输入服务器响应，将其回送给客户端

5.3　实际的Web服务器会做些什么

例 5-1 显示的 Perl 服务器是一个 Web 服务器的小例子。最先进的商用 Web 服务器要比它复杂得多，但它们确实执行了几项同样的任务，如图 5-3 所示。

(1) 建立连接——接受一个客户端连接，或者如果不希望与这个客户端建立连接，就将其关闭。

(2) 接收请求——从网络中读取一条 HTTP 请求报文。

(3) 处理请求——对请求报文进行解释，并采取行动。

(4) 访问资源——访问报文中指定的资源。

(5) 构建响应——创建带有正确首部的 HTTP 响应报文。

(6) 发送响应——将响应回送给客户端。

(7) 记录事务处理过程——将与已完成事务有关的内容记录在一个日志文件中。 113

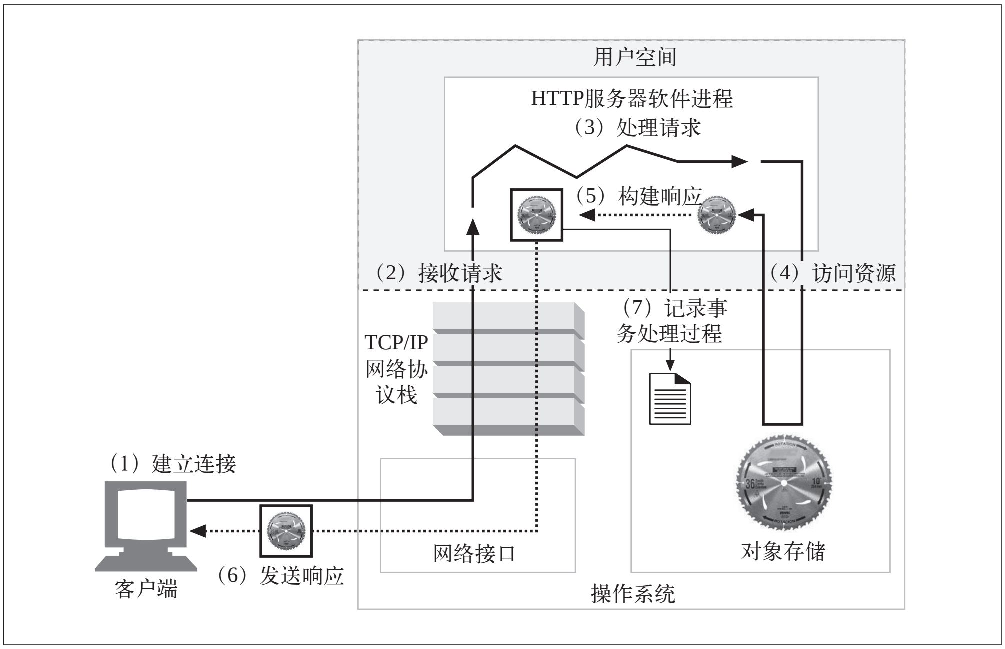


图 5-3　基本 Web 服务器请求的步骤 114

接下来的 7 个小节重点说明了 Web 服务器是怎样实现这些基本任务的。

5.4　第一步——接受客户端连接

如果客户端已经打开了一条到服务器的持久连接，可以使用那条连接来发送它的请求。否则，客户端需要打开一条新的到服务器的连接（回顾第 4 章，复习一下 HTTP 的连接管理技术）。

5.4.1　处理新连接

客户端请求一条到 Web 服务器的 TCP 连接时，Web 服务器会建立连接，判断连接的另一端是哪个客户端，从 TCP 连接中将 IP 地址解析出来。[1] 一旦新连接建立起来

并被接受，服务器就会将新连接添加到其现存 Web 服务器连接列表中，做好监视连接上数据传输的准备。

Web 服务器可以随意拒绝或立即关闭任意一条连接。有些 Web 服务器会因为客户端 IP 地址或主机名是未认证的，或者因为它是已知的恶意客户端而关闭连接。Web 服务器也可以使用其他识别技术。

5.4.2　客户端主机名识别

可以用“反向 DNS”对大部分 Web 服务器进行配置，以便将客户端 IP 地址转换成客户端主机名。Web 服务器可以将客户端主机名用于详细的访问控制和日志记录。但要注意的是，主机名查找可能会花费很长时间，这样会降低 Web 事务处理的速度。很多大容量 Web 服务器要么会禁止主机名解析，要么只允许对特定内容进行解析。

可以用配置指令 HostnameLookups 启用 Apache 的主机查找功能。比如，例 5-2 中的 Apache 配置指令就只打开了 HTML 和 CGI 资源的主机名解析功能。

例 5-2　配置 Apache，为 HTML 和 CGI 资源查找主机名

```
HostnameLookups off
<Files ~ "\.(html|htm|cgi)$">
    HostnameLookups on
</Files>
```

5.4.3　通过ident确定客户端用户

有些 Web 服务器还支持 IETF 的 ident 协议。服务器可以通过 ident 协议找到发起
115 HTTP 连接的用户名。这些信息对 Web 服务器的日志记录特别有用——流行的通用日志格式（Common Log Format）的第二个字段中就包含了每条 HTTP 请求的 ident 用户名。[2]

如果客户端支持 ident 协议，就在 TCP 端口 113 上监听 ident 请求。图 5-4 说明了 ident 协议是如何工作的。在图 5-4a 中，客户端打开了一条 HTTP 连接。然后，服务器打开自己到客户端 ident 服务器端口（113）的连接，发送一条简单的请求，询问与（由客户端和服务器端口号指定的）新连接相对应的用户名，并从客户端解析出包含用户名的响应。

注 1：不同的操作系统在对 TCP 连接进行操作时会使用不同的接口和数据结构。在 Unix 环境下，TCP 连接是由一个套接字表示的，可以用 getpeername 调用从套接字中获取客户端的 IP 地址。

注 2：这个通用日志格式的 ident 字段被称为“rfc931”，这是根据定义 ident 协议的过时 RFC 版本（更新过的 ident 规范记录在 RFC1413 中）命名的。

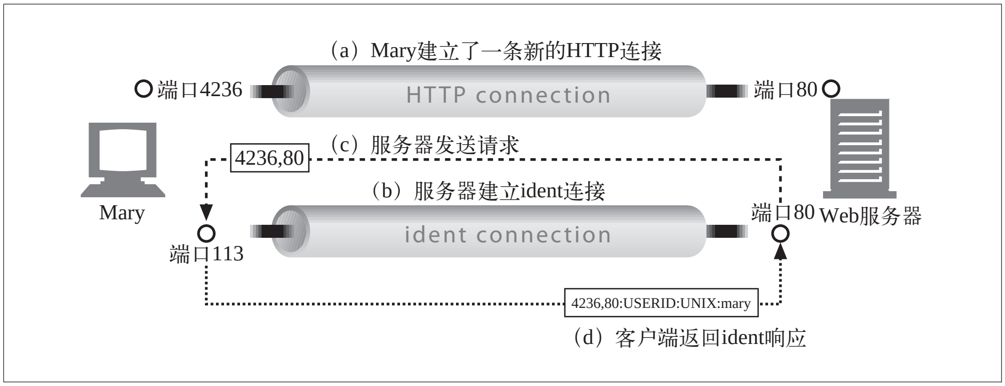


图 5-4　使用 ident 协议来确定 HTTP 的客户端用户名

ident 在组织内部可以很好地工作，但出于多种原因，在公共因特网上并不能很好地工作，原因包括：

- 很多客户端 PC 没有运行 ident 识别协议守护进程软件；
- ident 协议会使 HTTP 事务处理产生严重的时延；
- 很多防火墙不允许 ident 流量进入；
- ident 协议不安全，容易被伪造；
- ident 协议也不支持虚拟 IP 地址；
- 暴露客户端的用户名还涉及隐私问题。

可以通过 Apache 的 IdentityCheck on 指令告知 Apache Web 服务器使用 ident 查找功能。如果没有 ident 信息可用，Apache 会用连字符（-）来填充 ident 日志字段。由于没有 ident 信息可用，在使用通用日志格式的日志文件中，第二个字段通常都是连字符。

5.5　第二步——接收请求报文

连接上有数据到达时，Web 服务器会从网络连接中读取数据，并将请求报文中的内容解析出来（参见图 5-5）。 116

解析请求报文时，Web 服务器会：

- 解析请求行，查找请求方法、指定的资源标识符（URI）以及版本号，[3] 各项之间由一个空格分隔，并以一个回车换行（CRLF）序列作为行的结束；[4]

注 3：HTTP 的初始版本 HTTP/0.9 并不支持版本号。有些 Web 服务器也支持没有版本号的情况，会将报文作为 HTTP/0.9 请求进行解析。

注 4：很多客户端会错误地将 LF 作为行结束的终止符发送，所以很多 Web 服务器都支持将 LF 或 CRLF 作为行结束序列使用。

- 读取以 CRLF 结尾的报文首部；
- 检测到以 CRLF 结尾的、标识首部结束的空行（如果有的话）；
- 如果有的话（长度由 `Content-Length` 首部指定），读取请求主体。

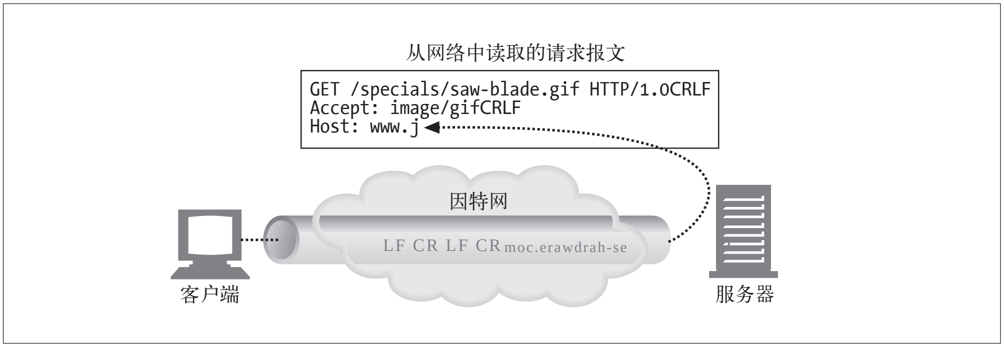


图 5-5　从连接中读取请求报文

解析请求报文时，Web 服务器会不定期地从网络上接收输入数据。网络连接可能随时都会出现延迟。Web 服务器需要从网络中读取数据，将部分报文数据临时存储在内存中，直到收到足以进行解析的数据并理解其意义为止。

5.5.1　报文的内部表示法

有些 Web 服务器还会用便于进行报文操作的内部数据结构来存储请求报文。比如，数据结构中可能包含有指向请求报文中各个片段的指针及其长度，这样就可以将这些首部存放在一个快速查询表中，以便快速访问特定首部的具体值了（参见图 5-6）。

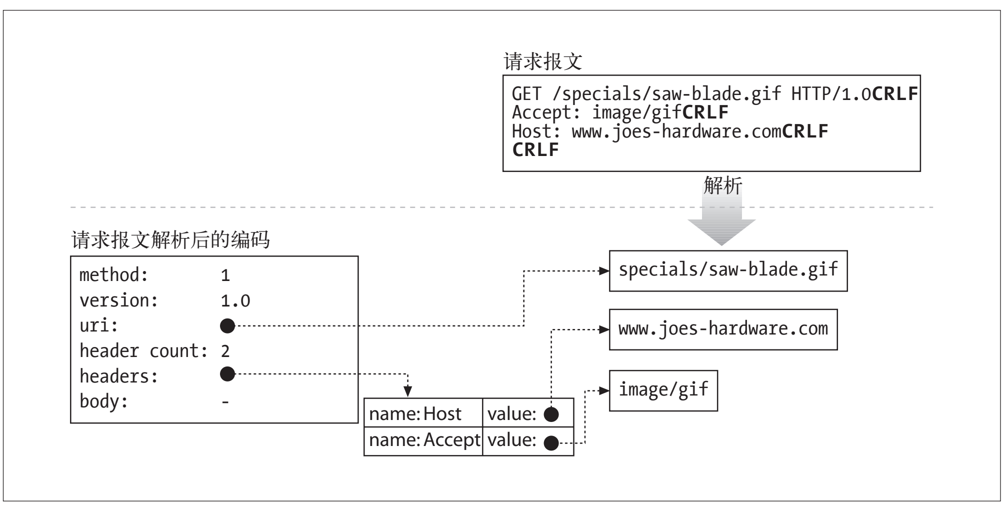


图 5-6　将请求报文解析为便捷的内部表示形式

5.5.2 连接的输入/输出处理结构

高性能的 Web 服务器能够同时支持数千条连接。这些连接使得服务器可以与世界各地的客户端进行通信，每个客户端都向服务器打开了一条或多条连接。某些连接可能在快速地向 Web 服务器发送请求，而其他一些连接则可能在慢慢发送，或者不经常发送请求，还有一些可能是空闲的，安静地等待着将来可能出现的动作。 117

因为请求可能会在任意时刻到达，所以 Web 服务器会不停地观察有无新的 Web 请求。不同的 Web 服务器结构会以不同的方式为请求服务，如图 5-7 所示。

- 单线程 Web 服务器（参见图 5-7a）

  单线程的 Web 服务器一次只处理一个请求，直到其完成为止。一个事务处理结束之后，才去处理下一条连接。这种结构易于实现，但在处理过程中，所有其他连接都会被忽略。这样会造成严重的性能问题，只适用于低负荷的服务器，以及 type-o-serve 这样的诊断工具。

- 多进程及多线程 Web 服务器（参见图 5-7b）

  多进程和多线程 Web 服务器用多个进程,或更高效的线程同时对请求进行处理。[5] 可以根据需要创建，或者预先创建一些线程 / 进程。[6] 有些服务器会为每条连接分配一个线程 / 进程，但当服务器同时要处理成百、上千，甚至数以万计的连接时，需要的进程或线程数量可能会消耗太多的内存或系统资源。因此，很多多线程 Web 服务器都会对线程 / 进程的最大数量进行限制。 118

- 复用 I/O 的服务器（参见图 5-7c）

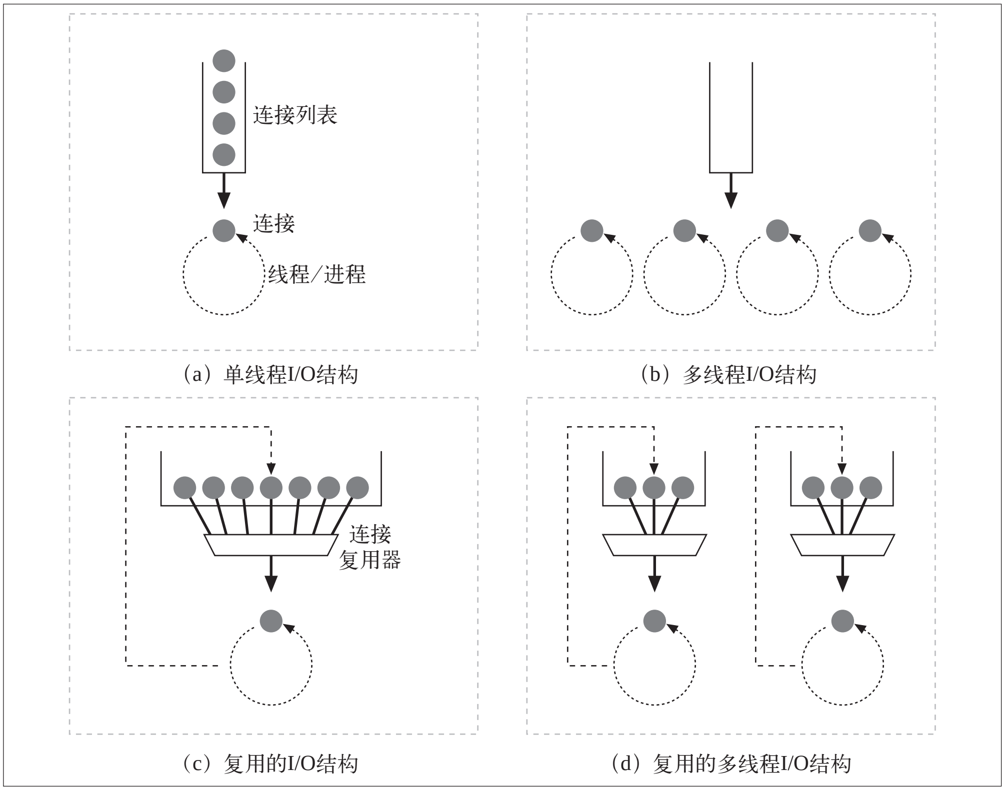
  为了支持大量的连接，很多 Web 服务器都采用了复用结构。在复用结构中，要同时监视所有连接上的活动。当连接的状态发生变化时（比如，有数据可用，或出现错误时），就对那条连接进行少量的处理；处理结束之后，将连接返回到开放连接列表中，等待下一次状态变化。只有在有事情可做时才会对连接进行处理；在空闲连接上等待的时候并不会绑定线程和进程。

- 复用的多线程 Web 服务器（参见图 5-7d）

  有些系统会将多线程和复用功能结合在一起，以利用计算机平台上的多个 CPU。多个线程（通常是一个物理处理器）中的每一个都在观察打开的连接（或打开的连接中的一个子集），并对每条连接执行少量的任务。

注 5：进程是一个独立的程序控制流，有自己的变量集。线程是一种更快、更高效的进程版本。单个程序可以通过线程和进程同时处理多件事情。为了便于解释，我们将线程和进程当作是可以互换的概念。但由于性能的不同，很多高性能服务器既是多进程的，又是多线程的。

注 6：会预先创建一些线程的系统被称为“工作池”系统，因为池中会有一组线程在等待工作。



119 图 5-7　Web 服务器输入 / 输出结构

5.6　第三步——处理请求

一旦 Web 服务器收到了请求，就可以根据方法、资源、首部和可选的主体部分来对请求进行处理了。

有些方法（比如 POST）要求请求报文中必须带有实体主体部分的数据。其他一些方法（比如 OPTIONS）允许有请求的主体部分，也允许没有。少数方法（比如 GET）禁止在请求报文中包含实体的主体数据。

这里我们并不对请求的具体处理方式进行讨论，因为本书其余大多数章节都在讨论这个问题。

5.7　第四步——对资源的映射及访问

Web 服务器是资源服务器。它们负责发送预先创建好的内容，比如 HTML 页面或 JPEG 图片，以及运行在服务器上的资源生成程序所产生的动态内容。

在 Web 服务器将内容传送给客户端之前，要将请求报文中的 URI 映射为 Web 服务器上适当的内容或内容生成器，以识别出内容的源头。

5.7.1 docroot

Web 服务器支持各种不同类型的资源映射，但最简单的资源映射形式就是用请求 URI 作为名字来访问 Web 服务器文件系统中的文件。通常，Web 服务器的文件系统中会有一个特殊的文件夹专门用于存放 Web 内容。这个文件夹被称为文档的根目录（document root，或 docroot）。Web 服务器从请求报文中获取 URI，并将其附加在文档根目录的后面。

在图 5-8 中，有一条对 /specials/saw-blade.gif 的请求到达。这个例子中 Web 服务器的文档根目录为 /usr/local/httpd/files。Web 服务器会返回文件 /usr/local/httpd/files/specials/saw-blade.gif。

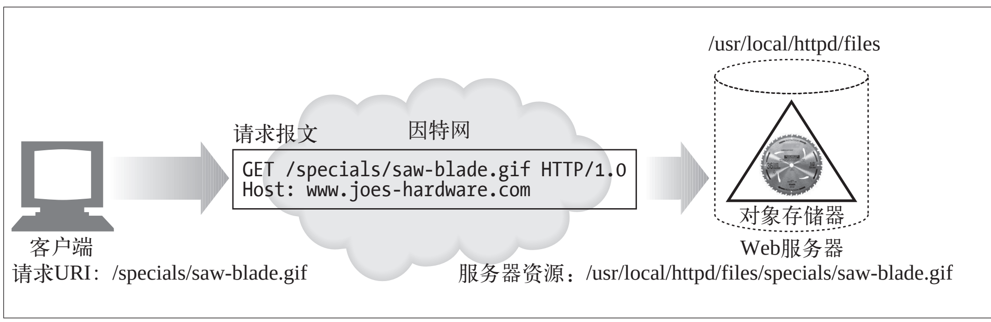


图 5-8　将请求 URI 映射为本地 Web 服务器上的资源 120

在配置文件 httpd.conf 中添加一个 `DocumentRoot` 行就可以为 Apache Web 服务器设置文档的根目录了：

```
DocumentRoot /usr/local/httpd/files
```

服务器要注意，不能让相对 URL 退到 docroot 之外，将文件系统的其余部分暴露出来。比如，大多数成熟的 Web 服务器都不允许这样的 URI 看到 Joe 的五金商店文档根目录上一级的文件：

http://www.joes-hardware.com/../

1. 虚拟托管的docroot

虚拟托管的 Web 服务器会在同一台 Web 服务器上提供多个 Web 站点，每个站点在服务器上都有自己独有的文档根目录。虚拟托管 Web 服务器会根据 URI 或 `Host` 首

部的 IP 地址或主机名来识别要使用的正确文档根目录。通过这种方式，即使请求 URI 完全相同，托管在同一 Web 服务器上的两个 Web 站点也可以拥有完全不同的内容了。

图 5-9 中的服务器托管了两个站点：www.joes-hardware.com 和 www.marys-antiques.com。服务器可以通过 HTTP 的 `Host` 首部，或根据不同的 IP 地址来区分不同的 Web 站点。

- 当请求 A 到达时，服务器会获取文件 /docs/joe/index.html。
- 当请求 B 到达时，服务器会获取文件 /docs/mary/index.html。

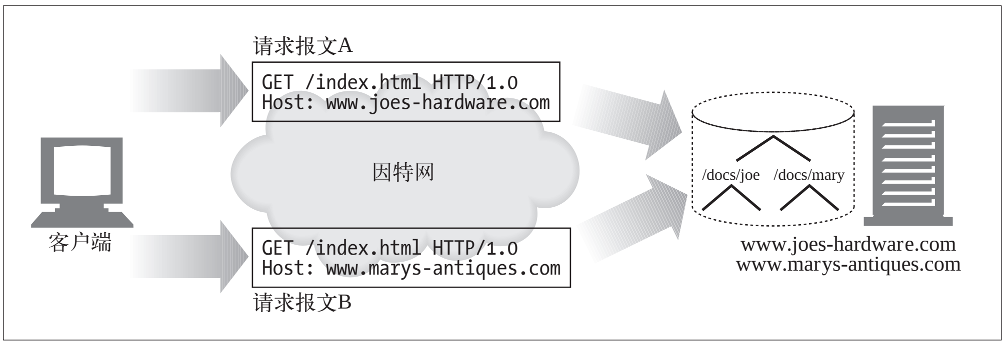


图 5-9　虚拟托管的请求会使用不同的文档根目录

对大多数 Web 服务器来说，配置虚拟托管的文档根目录是很简单的。对常见的 Apache Web 服务器来说，需要为每个虚拟 Web 站点配置一个 VirtualHost 块，而且每个虚拟服务器都要包含 `DocumentRoot`（例 5-3）。

例 5-3　Apache Web 服务器虚拟主机的 docroot 配置

121

```
<VirtualHost www.joes-hardware.com>
  ServerName www.joes-hardware.com
  DocumentRoot /docs/joe
  TransferLog /logs/joe.access_log
  ErrorLog /logs/joe.error_log
</VirtualHost>

<VirtualHost www.marys-antiques.com>
  ServerName www.marys-antiques.com
  DocumentRoot /docs/mary
  TransferLog /logs/mary.access_log
  ErrorLog /logs/mary.error_log
</VirtualHost>
    ...
```

更多与虚拟托管有关的信息可以参考 18.2 节。

2. 用户的主目录docroot

Docroot 的另一种常见应用是在 Web 服务器上为人们提供私有的 Web 站点。通常会把那些以斜杠和波浪号（/~）开始，后面跟着用户名的 URI 映射为此用户的私有文档根目录。私有 docroot 通常都是用户主目录下那个名为 public_html 的目录，但也可将其配置为其他值（参见图 5-10）。

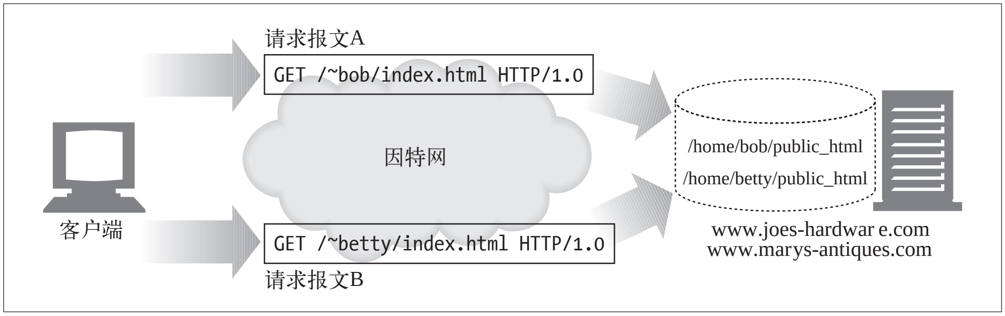


图 5-10　不同用户有不同的 docroot

5.7.2　目录列表

Web 服务器可以接收对目录 URL 的请求，其路径可以解析为一个目录，而不是文件。我们可以对大多数 Web 服务器进行配置，使其在客户端请求目录 URL 时采取不同的动作。

- 返回一个错误。
- 不返回目录，返回一个特殊的默认“索引文件”。
- 扫描目录，返回一个包含目录内容的 HTML 页面。

大多数 Web 服务器都会去查找目录中一个名为 index.html 或 index.htm 的文件来代表此目录。如果用户请求的是一个目录的 URL，而且这个目录中有一个名为 index.html（或 index.htm）的文件，服务器就会返回那个文件的内容。 122

在 Apache Web 服务器上，可以用配置指令 `DirectoryIndex` 来配置要作为默认目录文件使用的文件名集合。指令 `DirectoryIndex` 会按照优先顺序列出所有可以作为目录索引文件使用的文件名。下列配置行会使 Apache 在收到一个目录 URL 请求时，在目录中搜索命令中列出来的任意一个文件：

```
DirectoryIndex index.html index.htm home.html home.htm index.cgi
```

如果用户请求目录 URI 时，没有提供默认的索引文件，而且没有禁止使用目录索引，很多 Web 服务器都会自动返回一个 HTML 文件，此文件中会列出那个目录里

的文件名，以及每个文件的大小和修改日期，还包括到每个文件的 URI 链接。使用这个文件列表可能会很方便，但有些好事者也可以通过它在 Web 服务器上找到一些通常找不到的东西。

可以通过以下 Apache 指令禁止自动生成目录索引文件：

```
Options -Indexes
```

5.7.3 动态内容资源的映射

Web 服务器还可以将 URI 映射为动态资源——也就是说，映射到按需动态生成内容的程序上去（参见图 5-11）。实际上，有一大类名为应用程序服务器的 Web 服务器会将 Web 服务器连接到复杂的后端应用程序上去。Web 服务器要能够分辨出资源什么时候是动态的，动态内容生成程序位于何处，以及如何运行那个程序。大多数 Web 服务器都提供了一些基本的机制以识别和映射动态资源。

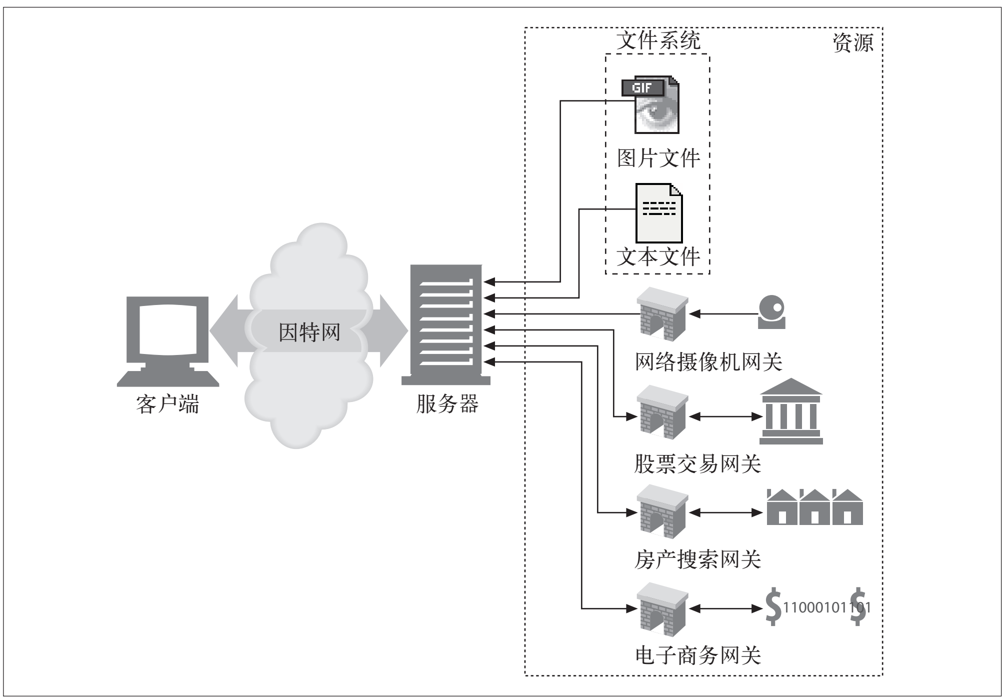


图 5-11　Web 服务器可以提供静态资源和动态资源

Apache 允许用户将 URI 路径名组件映射为可执行文件目录。服务器收到一条带有可执行路径组件的对 URI 的请求时，会试着去执行相应服务器目录中的程序。例如，下列 Apache 配置指令就表明所有路径以 /cgi-bin/ 开头的 URI 都应该执行在目录 /usr/local/etc/httpd/cgi-programs/ 中找到的相应文件：

```
ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-programs/
```

Apache 还允许用户用一个特殊的文件扩展名来标识可执行文件。通过这种方式就可以将可执行脚本放在任意目录中了。下面的 Apache 配置指令说明要执行所有以 .cgi 结尾的 Web 资源：

```
AddHandler cgi-script .cgi
```

CGI 是早期出现的一种简单、流行的服务端应用程序执行接口。现代的应用程序服务器都有更强大更有效的服务端动态内容支持机制，包括微软的动态服务器页面（Active Server Page）和 Java servlet。 123

5.7.4 服务器端包含项

很多 Web 服务器还提供了对服务器端包含项（SSI）的支持。如果某个资源被标识为存在服务器端包含项，服务器就会在将其发送给客户端之前对资源内容进行处理。

要对内容进行扫描，以查找（通常包含在特定 HTML 注释中的）特定的模板，这些模板可以是变量名，也可以是嵌入式脚本。可以用变量的值或可执行脚本的输出来取代特定的模板。这是创建动态内容的一种简便方式。

5.7.5 访问控制

Web 服务器还可以为特定资源进行访问控制。有请求到达，要访问受控资源时，Web 服务器可以根据客户端的 IP 地址进行访问控制，也可以要求输入密码来访问资源。

更多与 HTTP 认证有关的信息请参见第 12 章。 124

5.8 第五步——构建响应

一旦 Web 服务器识别出了资源，就执行请求方法中描述的动作，并返回响应报文。响应报文中包含有响应状态码、响应首部，如果生成了响应主体的话，还包括响应主体。3.4 节详细介绍了 HTTP 响应代码。

5.8.1 响应实体

如果事务处理产生了响应主体，就将内容放在响应报文中回送过去。如果有响应主体的话，响应报文中通常包括：

- 描述了响应主体 MIME 类型的 `Content-Type` 首部；

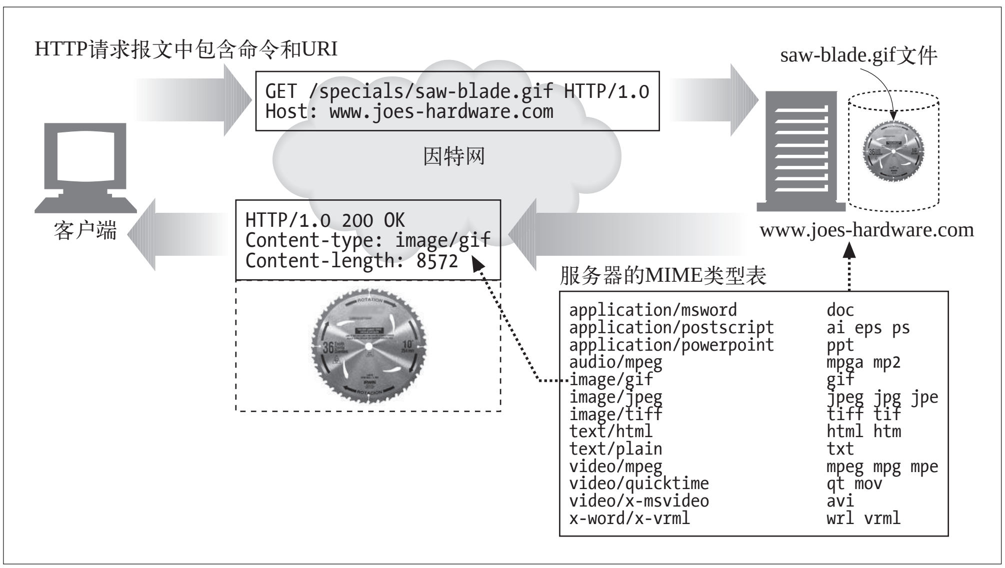
- 描述了响应主体长度的 `Content-Length` 首部；
- 实际报文的主体内容。

5.8.2 MIME类型

Web 服务器要负责确定响应主体的 MIME 类型。有很多配置服务器的方法可以将 MIME 类型与资源关联起来。

- MIME 类型（mime.types）

  Web 服务器可以用文件的扩展名来说明 MIME 类型。Web 服务器会为每个资源扫描一个包含了所有扩展名的 MIME 类型的文件，以确定其 MIME 类型。这种基于扩展名的类型相关是最常见的，参见图 5-12。



125 图 5-12 Web 服务器用 MIME 类型文件来设置资源输出的 `Content-type` 首部

- 魔法分类（Magic typing）

  Apache Web 服务器可以扫描每个资源的内容，并将其与一个已知模式表（被称为魔法文件）进行匹配，以决定每个文件的 MIME 类型。这样做可能比较慢，但很方便，尤其是文件没有标准扩展名的时候。

- 显式分类（Explicit typing）

  可以对 Web 服务器进行配置，使其不考虑文件的扩展名或内容，强制特定文件或目录内容拥有某个 MIME 类型。

- 类型协商

  有些 Web 服务器经过配置，可以以多种文档格式来存储资源。在这种情况下，可以配置 Web 服务器，使其可以通过与用户的协商来决定使用哪种格式（及相关的 MIME 类型）“最好”。这个问题将在第 17 章讨论。

  还可以通过配置 Web 服务器，将特定的文件与 MIME 类型相关联。

5.8.3 重定向

Web 服务器有时会返回重定向响应而不是成功的报文。 Web 服务器可以将浏览器重定向到其他地方来执行请求。重定向响应由返回码 3XX 说明。`Location` 响应首部包含了内容的新地址或优选地址的 URI。重定向可用于下列情况。

- 永久搬离的资源

  资源可能已经被移动到了新的位置，或者被重新命名，有了一个新的 URL。Web 服务器可以告诉客户端资源已经被重命名了，这样客户端就可以在从新地址获取资源之前，更新书签之类的信息了。状态码 301 Moved Permanently 就用于此类重定向。

- 临时搬离的资源

  如果资源被临时移走或重命名了，服务器可能希望将客户端重定向到新的位置上去。但由于重命名是临时的，所以服务器希望客户端将来还可以回头去使用老的 URL，不要对书签进行更新。状态码 303 See Other 以及状态码 307 Temporary Redirect 就用于此类重定向。

- URL 增强

  服务器通常用重定向来重写 URL，往往用于嵌入上下文。当请求到达时，服务器会生成一个新的包含了嵌入式状态信息的 URL，并将用户重定向到这个新的 URL 上去。[7] 客户端会跟随这个重定向信息，重新发起请求，但这次的请求会包含完整的、经过状态增强的 URL。这是在事务间维护状态的一种有效方式。状态码 303 See Other 和 307 Temporary Redirect 用于此类重定向。 126

- 负载均衡

  如果一个超载的服务器收到一条请求，服务器可以将客户端重定向到一个负载不太重的服务器上去。状态码 303 See Other 和 307 Temporary Redirect 可用于此类重定向。

- 服务器关联

  Web 服务器上可能会有某些用户的本地信息；服务器可以将客户端重定向到包含了那个客户端信息的服务器上去。状态码 303 See Other 和 307 Temporary Redirect 可用于此类重定向。

注 7：有时会将这些经过扩展和状态增强的 URL 称为“胖 URL”。

- 规范目录名称
  客户端请求的 URI 是一个不带尾部斜线的目录名时，大多数 Web 服务器都会将客户端重定向到一个加了斜线的 URI 上，这样相对链接就可以正常工作了。

5.9 第六步——发送响应

Web 服务器通过连接发送数据时也会面临与接收数据一样的问题。服务器可能有很多条到各个客户端的连接，有些是空闲的，有些在向服务器发送数据，还有一些在向客户端回送响应数据。

服务器要记录连接的状态，还要特别注意对持久连接的处理。对非持久连接而言，服务器应该在发送了整条报文之后，关闭自己这一端的连接。

对持久连接来说，连接可能仍保持打开状态，在这种情况下，服务器要特别小心，要正确地计算 `Content-Length` 首部，不然客户端就无法知道响应什么时候结束了（参见第 4 章）。

5.10 第七步——记录日志

最后，当事务结束时，Web 服务器会在日志文件中添加一个条目，来描述已执行的事务。大多数 Web 服务器都提供了几种日志配置格式。更多细节请参见第 21 章。

5.11 更多信息

更多有关 Apache、Jigsaw 和 ident 的信息，参见以下参考资源

- *Apache:The Definitive Guide*[8]（《Apache 权威指南》）
  Ben Laurie 和 Peter Laurie 编写，O'Reilly & Associates 公司出版。

- *Professional Apache*
  127 Peter Wainwright 编写，Wrox 公司出版。

- http://www.w3c.org/Jigsaw/
  Jigsaw——W3C 的服务器 W3C 联盟 Web 站点。

- http://www.ietf.org/rfc/rfc1413.txt
  128 RFC 1413，M. St. Johns 编写的“Identification Protocol”（“标识协议”）。

注 8：本书影印版由中国电力出版社出版。（编者注）

第6章

代理

Web 代理（proxy）服务器是网络的中间实体。代理位于客户端和服务器之间，扮演“中间人”的角色，在各端点之间来回传送 HTTP 报文。本章介绍了所有与 HTTP 代理服务器有关的内容，为代理特性提供的特殊支持，以及使用代理服务器时会遇到的一些棘手的问题。

本章主要内容如下：

- 对 HTTP 代理进行解释，将其与 Web 网关进行对比，并说明如何部署代理；
- 给出一些代理所能提供的帮助；
- 说明在现实网络中是怎样部署代理以及如何将网络流量导向代理服务器；
- 说明如何配置浏览器来使用代理；
- 展示 HTTP 的代理请求，说明它们与服务器请求的区别，以及代理是如何微妙地改变浏览器行为的；
- 解释如何通过 `Via` 首部和 TRACE 方法来记录报文传输路径上的代理服务器链；
- 描述基于代理的 HTTP 访问控制方法；
- 解释代理如何与客户端和服务器进行交互，每个客户端和服务器支持的特性和使用的版本都可能有所不同。

6.1 Web的中间实体

Web 上的代理服务器是代表客户端完成事务处理的中间人。如果没有 Web 代理，HTTP 客户端就要直接与 HTTP 服务器进行对话。有了 Web 代理，客户端就可以与代理进行对话，然后由代理代表客户端与服务器进行交流。客户端仍然会完成对事
129 务的处理，但它是通过代理服务器提供的优质服务来实现的。

HTTP 的代理服务器既是 Web 服务器又是 Web 客户端。HTTP 客户端会向代理发送请求报文，代理服务器必须像 Web 服务器一样，正确地处理请求和连接，然后返回响应。同时，代理自身要向服务器发送请求，这样，其行为就必须像正确的 HTTP 客户端一样，要发送请求并接收响应（参见图 6-1）。如果要创建自己的 HTTP 代理，就要认真地遵循为 HTTP 客户端和 HTTP 服务器制定的规则。

6.1.1 私有和共享代理

代理服务器可以是某个客户端专用的，也可以是很多客户端共享的。单个客户端专用的代理被称为*私有代理*。众多客户端共享的代理被称为*公共代理*。

- *公共代理*

  大多数代理都是公共的共享代理。集中式代理的成本效率更高，更容易管理。某些

代理应用，比如高速缓存代理服务器，会利用用户间共同的请求，这样的话，汇入同一个代理服务器的用户越多，它就越有用。

- *私有代理*

  专用的私有代理并不常见，但它们确实存在，尤其是直接运行在客户端计算机上的时候。有些浏览器辅助产品，以及一些 ISP 服务，会在用户的 PC 上直接运行一些小型的代理，以便扩展浏览器特性，提高性能，或为免费 ISP 服务提供主机广告。

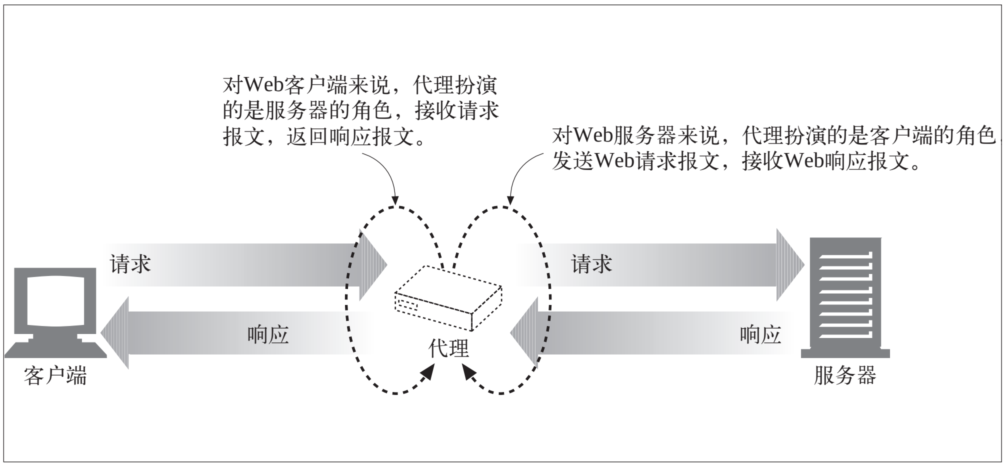


图 6-1　代理既是服务器，又是客户端

6.1.2　代理与网关的对比

严格来说，代理连接的是两个或多个使用相同协议的应用程序，而网关连接的则是两个或多个使用不同协议的端点。网关扮演的是“协议转换器”的角色，即使客户端和服务器使用的是不同的协议，客户端也可以通过它完成与服务器之间的事务处理。 130

图 6-2 显示了代理和网关之间的区别。

- 图 6-2a 中的中间设备是一个 HTTP 代理，因为代理与客户端和服务器之间使用的都是 HTTP 协议。
- 图 6-2b 中的中间设备是一个 HTTP/POP 网关，因为它把 HTTP 的前台与 POP E-mail 的后端连接了起来。网关将 Web 事务转换成适当的 POP 事务，这样用户就可以通过 HTTP 读取 E-mail 了。基于 Web 的 E-mail 程序，比如 Yahoo! 邮件和 MSN Hotmail 都是 HTTP E-mail 网关。

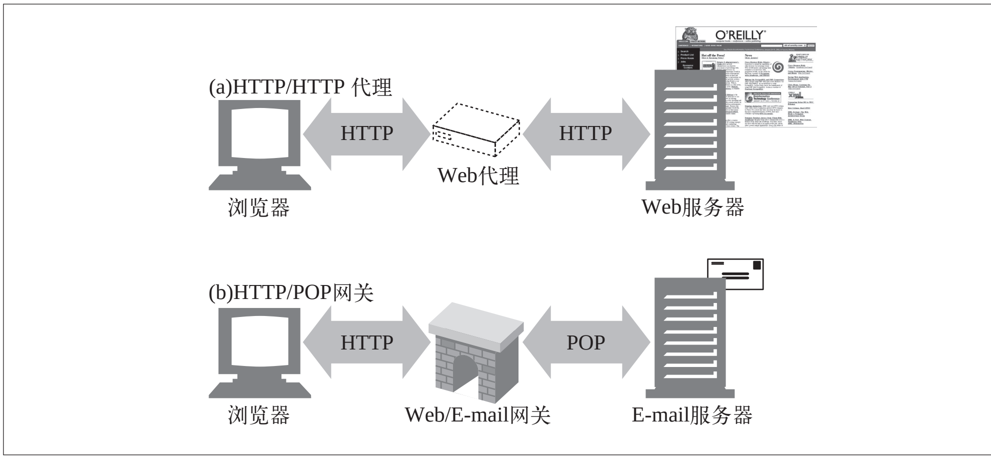


图 6-2　代理使用同一种协议，网关则将不同的协议连接起来

实际上，代理和网关之间的区别很模糊。由于浏览器和服务器实现的是不同版本的 HTTP，代理也经常要做一些协议转换工作。而商业化的代理服务器也会实现网关的功能来支持 SSL 安全协议、SOCKS 防火墙、FTP 访问，以及基于 Web 的应用程序。我们将在第 8 章详细介绍网关。

6.2　为什么使用代理

代理服务器可以实现各种时髦且有用的功能。它们可以改善安全性，提高性能，节省费用。代理服务器可以看到并接触到所有流过的 HTTP 流量，所以代理可以监视流量并对其进行修改，以实现很多有用的增值 Web 服务。这里给出了几种代理使用方法的示例。

- *儿童过滤器*

  小学在为教育站点提供无阻碍访问的同时，可以利用过滤器代理来阻止学生访问
  131 成人内容。如图 6-3 所示，代理应该允许学生无限制地访问教育性内容，但对不适合儿童的站点要强行禁止访问。[1]

- *文档访问控制*

  可以用代理服务器在大量 Web 服务器和 Web 资源之间实现统一的访问控制策略，创建审核跟踪机制。这在大型企业环境或其他分布式机构中是很有用的。

  在集中式代理服务器上可以对所有访问控制功能进行配置，而无需在众多由不同组织管理、不同厂商制造、使用不同模式的 Web 服务器上进行经常性

注 1：有些公司和非营利性组织提供了过滤软件，维护了一些“黑名单”，以识别并限制对危害性内容的访问。

的访问控制升级。[2]

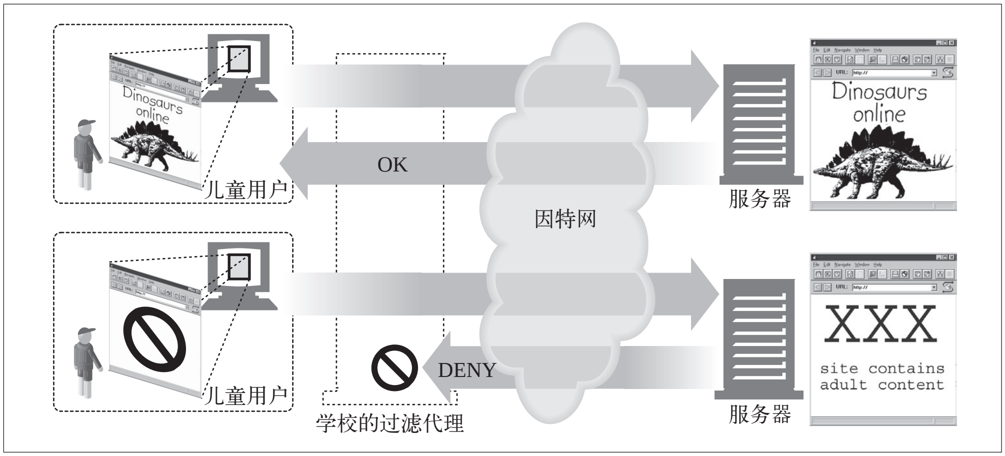


图 6-3　代理应用程序实例：儿童安全的因特网过滤器

在图 6-4 中，集中式访问控制代理：

- 允许客户端 1 无限制地访问服务器 A 的新闻页面；
- 客户端 2 可以无限制地访问因特网；
- 在允许客户端 3 访问服务器 B 之前，要求其输入口令。

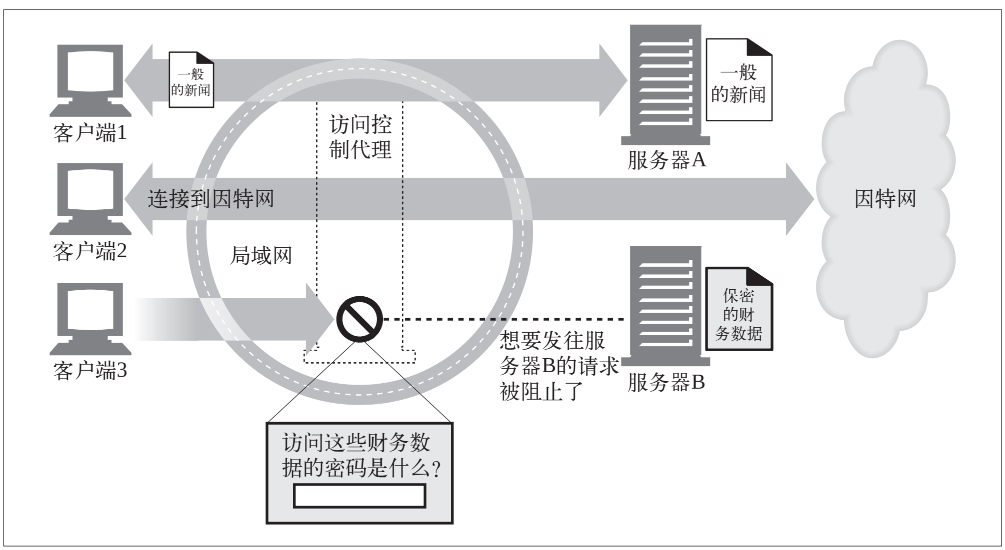


图 6-4　代理应用程序实例：集中式文档访问控制

注 2：为防止一些经验丰富的用户蓄意绕过控制代理，可以静态地配置 Web 服务器，使其仅接受来自代理服务器的请求。

- 安全防火墙

  网络安全工程师通常会使用代理服务器来提高安全性。代理服务器会在网络中的单一安全节点上限制哪些应用层协议的数据可以流入或流出一个组织。还可以提供用来消除病毒的 Web 和 E-mail 代理使用的那种挂钩程序，以便对流量进行详
132 细的检查（参见图 6-5）。

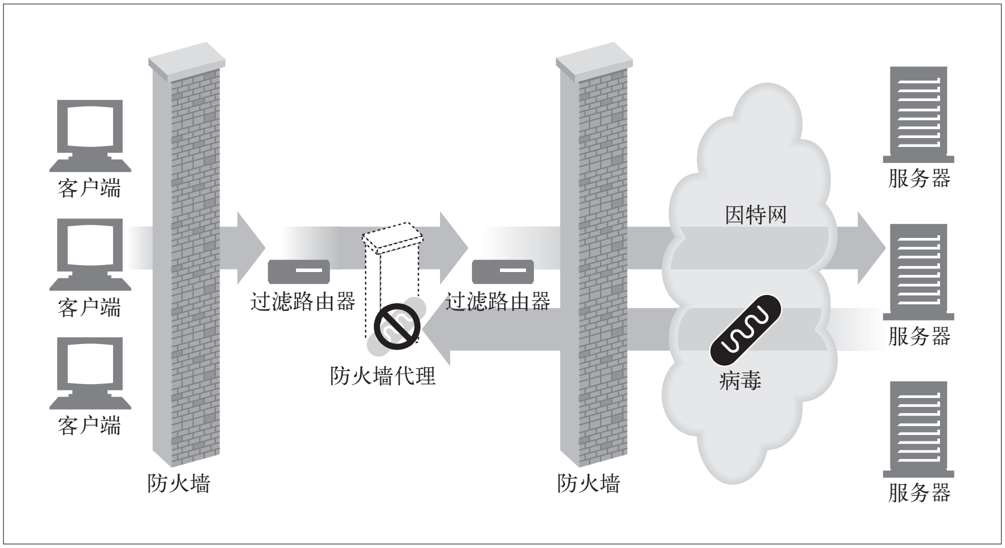


图 6-5　代理应用程序实例：安全防火墙

- Web 缓存

  代理缓存维护了常用文档的本地副本，并将它们按需提供，以减少缓慢且昂贵的因特网通信。

  在图 6-6 中，客户端 1 和客户端 2 会去访问附近 Web 缓存上的对象 A，而客户
133 端 3 和客户端 4 访问的则是原始服务器上的文档。

- 反向代理

  代理可以假扮 Web 服务器。这些被称为替代物（surrogate）或反向代理（reverse proxy）的代理接收发给 Web 服务器的真实请求，但与 Web 服务器不同的是，它们可以发起与其他服务器的通信，以便按需定位所请求的内容。

  可以用这些反向代理来提高访问慢速 Web 服务器上公共内容时的性能。在这种配置中，通常将这些反向代理称为服务器加速器（server accelerator）（参见图 6-7）。还可以将替代物与内容路由功能配合使用，以创建按需复制内容的分布式网络。

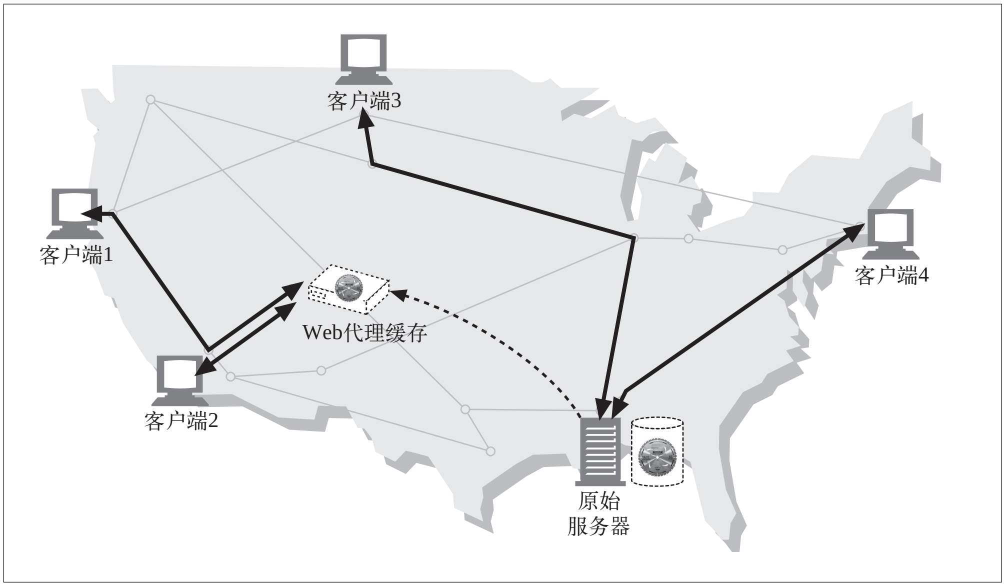


图 6-6　代理应用程序实例：Web 缓存

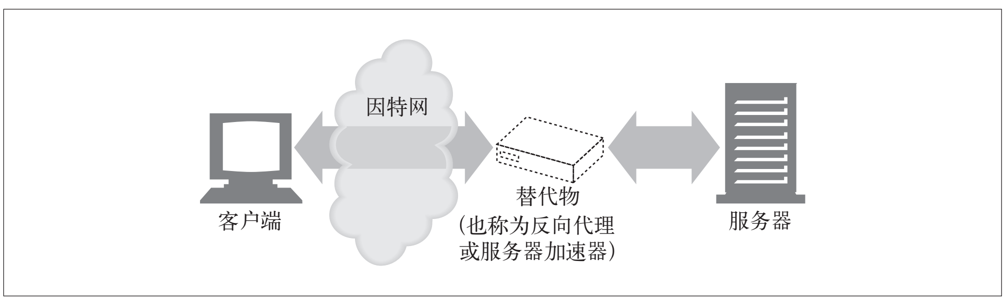


图 6-7　代理应用程序实例：（服务器加速器实现方式中的）替代物

- *内容路由器*

  代理服务器可以作为“内容路由器”使用，根据因特网流量状况以及内容类型将请求导向特定的 Web 服务器。

  内容路由器也可以用来实现各种服务级的请求。比如，如果用户或内容提供者付费要求提供更高的性能，内容路由器可以将请求转发到附近的复制缓存（参见图 6-8），或者如果用户申请了过滤服务，还可以通过过滤代理来转发 HTTP 请求。可以用自适应内容路由代理来构建很多有趣的服务。 134

- *转码器*

  代理服务器在将内容发送给客户端之前，可以修改内容的主体格式。在这些数据

表示法之间进行的透明转换被称为*转码*（transcoding）。[3]

转码代理可以在传输 GIF 图片时，将其转换成 JPEG 图片，以减小尺寸。也可以对图片进行压缩，或降低颜色的色彩饱和度以便在电视上观看。同样，可以对文本文件进行压缩，并为能够使用因特网的呼机和智能手机生成小型的文本摘要 Web 页面。代理甚至可以在传输文档的过程中将其转换成外语。

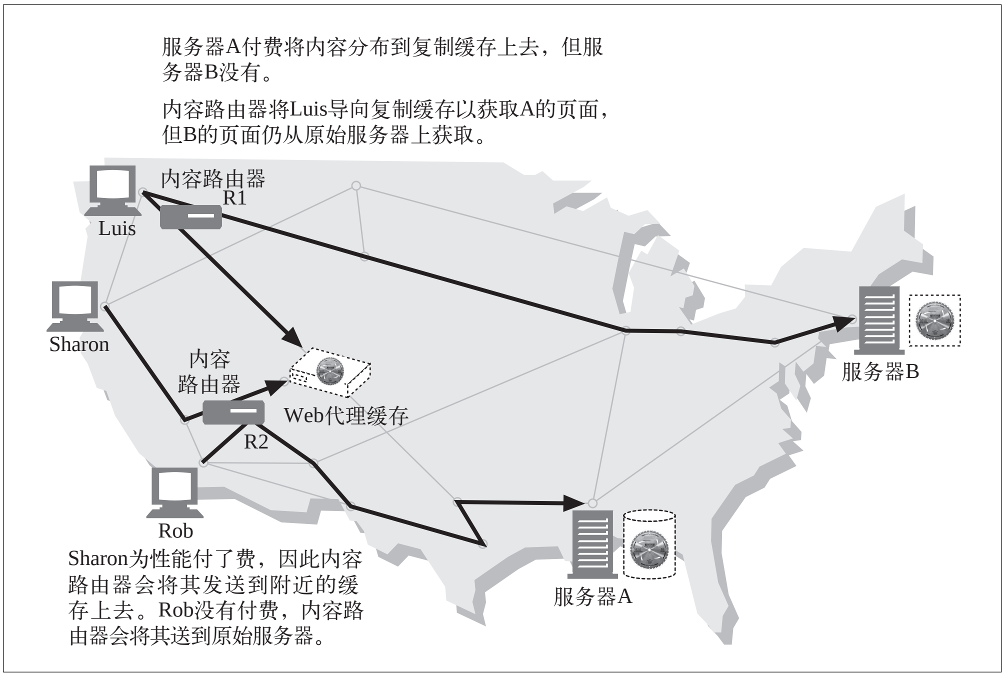


图 6-8　代理应用程序实例：内容路由器

图 6-9 显示了一个转码代理，这个代理可以将英语文本转换成西班牙语文本，将
135 HTML 页面重新格式化为较简单的文本，以便显示在手机的小屏幕上。

- *匿名者*

  匿名者代理会主动从 HTTP 报文中删除身份特性（比如客户端 IP 地址、`From` 首部、`Referer` 首部、cookie、URI 的会话 ID），从而提供高度的私密性和匿名性。[4]

注 3：有些人会对“转码”和“翻译”进行区分，将转码定义为对数据编码进行的相对简单的转换（比如无损压缩），将翻译定义为更重要的、对数据的重新格式化或语义转换。我们用术语转码来表示所有在中间实体上对内容进行的修改。

注 4：但是，由于删除了识别信息，用户浏览体验的质量可能会有所下降，有些 Web 站点可能会无法正常工作。

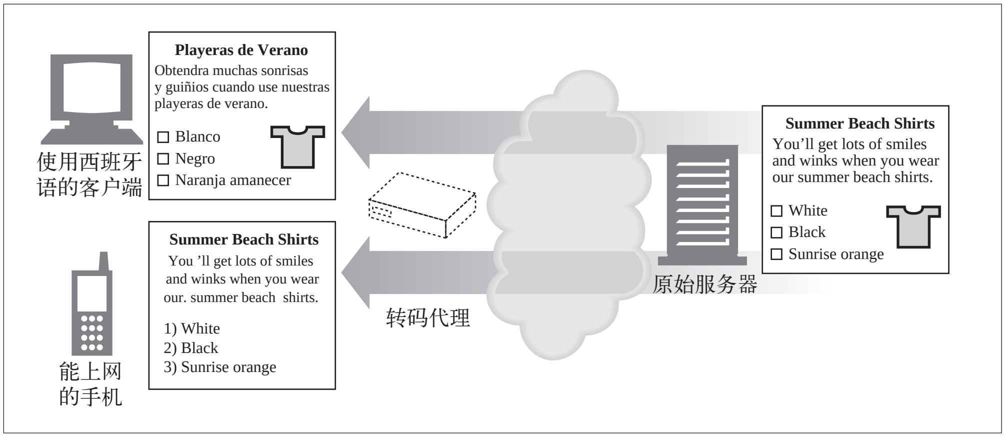


图 6-9　代理应用程序实例：内容转码器

在图 6-10 中，匿名代理会对用户报文进行下列修改以增加私密性。

- 从 `User-Agent` 首部删除用户的计算机与 OS 类型。
- 删除 `From` 首部以保护用户的 E-mail 地址。
- 删除 `Referer` 首部来掩盖用户访问过的其他站点。
- 删除 `Cookie` 首部以剔除概要信息和身份的数据。

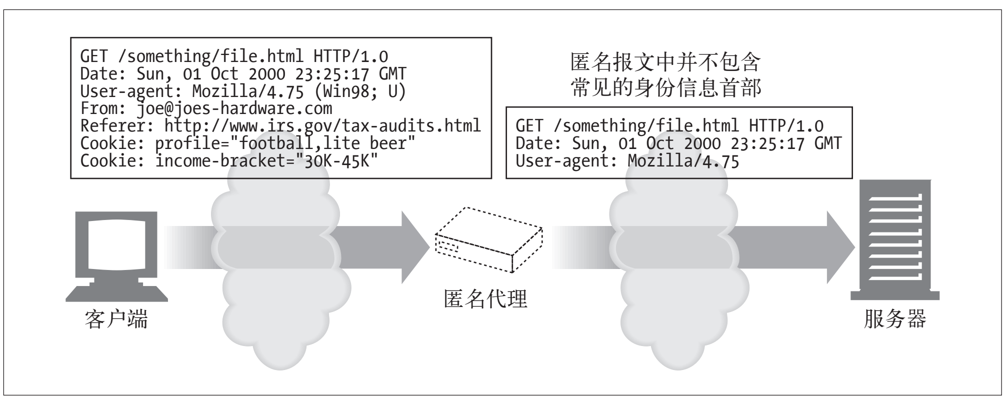


图 6-10　代理应用程序实例：匿名者 136

6.3　代理会去往何处

前一小节解释了代理会做些什么。现在我们来看看在一个网络结构中部署代理的时候，它会位于何处。本节会涵盖以下内容：

- 怎样将代理部署到网络中去；

- 怎样将代理以层级方式连接在一起；
- 怎样先将网络流量直接导入代理服务器中。

6.3.1 代理服务器的部署

可以根据其目标用途，将代理放在任意位置。图 6-11 给出了部署代理服务器的几种方式。

- 出口代理

  可以将代理固定在本地网络的出口点，以便控制本地网络与大型因特网之间的流量。可以在公司网络中使用出口代理（参见图 6-11a），提供针对公司外部恶意黑客的防火墙保护，或降低带宽费用，提高因特网流量的性能。小学可能会使用过滤出口代理来防止早熟的学生浏览不恰当的内容。

- 访问（入口）代理

  代理常被放在 ISP 访问点上，用以处理来自客户的聚合请求。ISP 使用缓存代理来存储常用文档的副本，以提高用户（尤其是高速连接用户）的下载速度，降低因特网带宽耗费（参见图 6-11b）。

- 反向代理

  代理通常会被部署在网络边缘，在 Web 服务器之前，作为替代物（也常被称为反向代理，参见图 6-11c）使用，在那里它们可以处理所有传送给 Web 服务器的请求，并只在必要时向 Web 服务器请求资源。替代物可以提高 Web 服务器的安全特性，或者将快速的 Web 服务器缓存放在较慢的服务器之前，以提高性能。反向代理通常会直接冒用 Web 服务器的名字和 IP 地址，这样所有的请求就会被发送给代理而不是服务器了。

- 网络交换代理

  可以将具有足够处理能力的代理放在网络之间的因特网对等交换点上，通过缓存
137 来减轻因特网节点的拥塞，并对流量进行监视，参见图 6-11d。[5]

6.3.2 代理的层次结构

可以通过代理层次结构（proxy hierarchy）将代理级联起来。如图 6-12 所示，在代理的层次结构中，会将报文从一个代理传给另一个代理，直到最终抵达原始服务器为止（然后通过代理传回给客户端）。

注 5：核心代理通常被部署在因特网带宽很昂贵的地方（尤其是在欧洲）。有些国家（比如英国）还会出于对国家安全的考虑，对有争议的代理部署进行评估，以监测因特网流量。

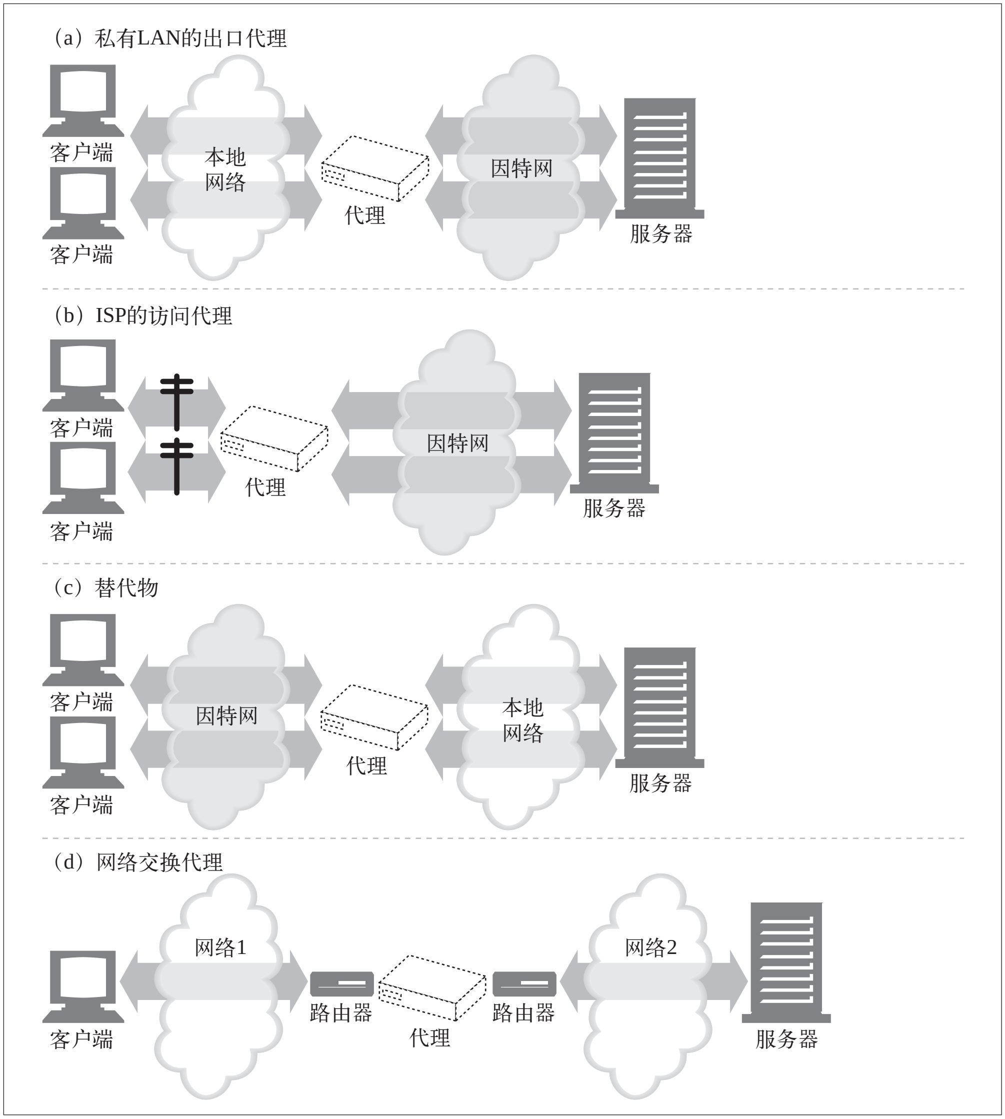


图 6-11　可以根据其目标用途，以多种方式来部署代理

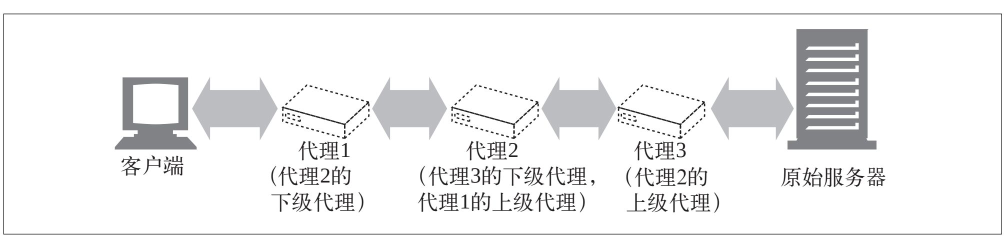


图 6-12　三级的代理层次结构

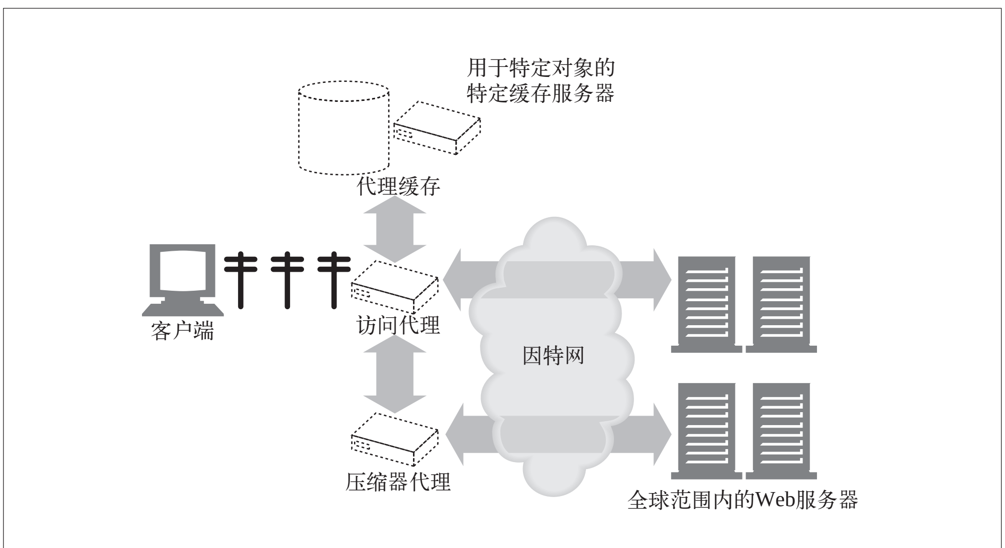
Proxy 层次结构中的代理服务器被赋予了父（parent）和子（child）的关系。下一个
138 入口（inbound）代理（靠近服务器）被称为父代理，下一个出口（outbound）代理（靠近客户端）被称为子代理。在图 6-12 中，代理 1 是代理 2 的子代理。同样，代理 2 是代理 3 的子代理，代理 3 是代理 2 的父代理。

代理层次结构的内容路由

图 6-12 中的代理层次结构是静态的——代理 1 总是会将报文转发给代理 2，代理 2 总是会将报文转发给代理 3。但是，层次不一定非得是静态的。代理服务器可以根据众多因素，将报文转发给一个不断变化的代理服务器和原始服务器集。

比如，在图 6-13 中，访问代理会根据不同的情况将报文转发给父代理或原始服务器。

- 如果所请求的对象属于一个付费使用内容分发服务的 Web 服务器，代理就会将请求发送给附近的一个缓存服务器，这个服务器会返回已缓存对象，或者如果它那儿没有的话，它会去取回内容。
- 如果请求的是特定类型的图片，访问代理会将请求转发给一个特定的压缩代理，这个代理会去获取图片，然后对其进行压缩，这样通过到客户端的慢速 Modem 下载时，速度会更快一些。



139 图 6-13　代理层次结构可以是动态的，随请求而变的

这里还有几个动态选择父代理的例子。

- *负载均衡*

  子代理可能会根据当前父代理上的工作负载级别来决定如何选择一个父代理，以均衡负载。

- *地理位置附近的路由*

  子代理可能会选择负责原始服务器所在物理区域的父代理。

- *协议 / 类型路由*

  子代理可能会根据 URI 将报文转发到不同的父代理和原始服务器上去。某些特定类型的 URI 可能要通过一些特殊的代理服务器转发请求，以便进行特殊的协议处理。

- *基于订购的路由*

  如果发布者为高性能服务额外付费了，它们的 URI 就会被转发到大型缓存或压缩引擎上去，以提高性能。

  在不同的产品中，动态父路由逻辑的实现方式各有不同，包括使用配置文件、脚本语言和动态可执行插件等。

6.3.3 代理是如何获取流量的

客户端通常会直接与 Web 服务器进行通信，所以我们要解释清楚 HTTP 流量怎样才能首先流向代理。有四种常见方式可以使客户端流量流向代理。

- *修改客户端*

  很多 Web 客户端，包括网景和微软的浏览器，都支持手工和自动的代理配置。如果将客户端配置为使用代理服务器，客户端就会将 HTTP 请求有意地直接发送给代理，而不是原始服务器（参见图 6-14a）。

- *修改网络*

  网络基础设施可以通过若干种技术手段，在客户端不知道，或没有参与的情况下，拦截网络流量并将其导入代理。这种拦截通常都依赖于监视 HTTP 流量的交换设备及路由设备，在客户端毫不知情的情况下，对其进行拦截，并将流量导入一个代理（参见图 6-14b）。这种代理被称为*拦截*（intercepting）代理。[6]

注 6：拦截代理通常被称为“透明代理”，因为你会在不知情的情况下连接到这些代理上去。但 HTTP 规范中已用“透明”来表示那些不会对语义进行修改的功能了，所以标准制定机构建议在流量捕获中使用术语“拦截”。这里采纳了这一术语。

- 修改 DNS 的命名空间

140 放在 Web 服务器之前的代理服务器——替代物，会直接假扮 Web 服务器的名字和 IP 地址，这样，所有的请求就会发送给这些替代物，而不是服务器了（参见图 6-14c）。要实现这一点，可以手工编辑 DNS 名称列表，或者用特殊的动态 DNS 服务器根据需要来确定适当的代理或服务器。有时在安装过程中，真实服务器的 IP 地址和名称被修改了，替代物得到的会是之前的地址和名称。

- 修改 Web 服务器

  也可以将某些 Web 服务器配置为向客户端发送一条 HTTP 重定向命令（响应码 305），将客户端请求重定向到一个代理上去。收到重定向命令后，客户端会与代理进行通信（参见图 6-14d）。

下一节解释了如何配置客户端才能使其将流量发送给代理。第 20 章会说明如何配置网络、DNS 以及服务器，才能将流量重定向到代理服务器。

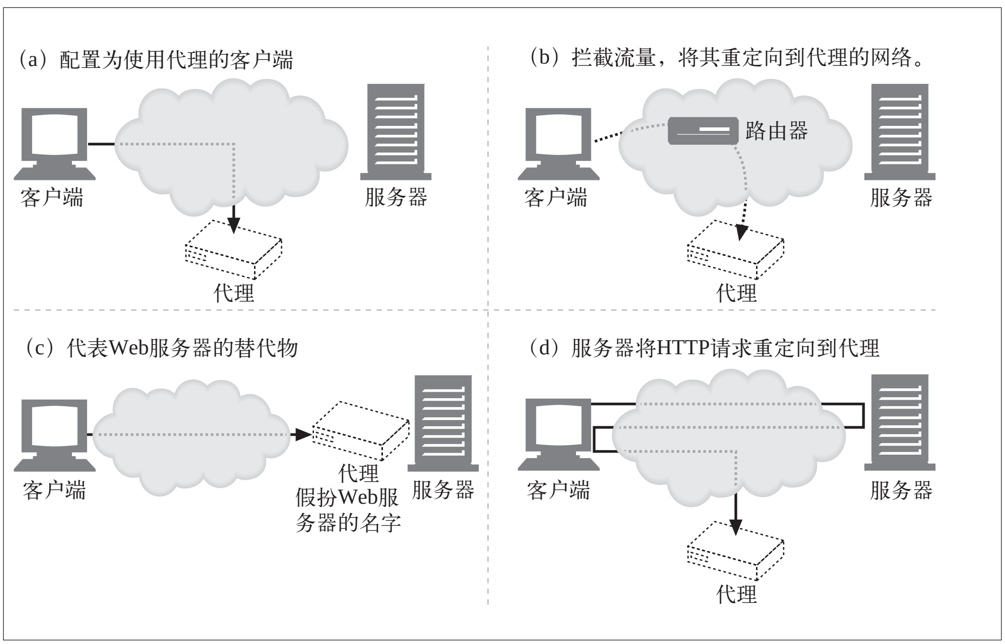


图 6-14　有很多技巧可以将 Web 请求导向代理

6.4　客户端的代理设置

所有现代的 Web 浏览器都允许用户对代理的使用进行配置。实际上，很多浏览器都提供了多种配置代理的方式，其中包括以下几种。

- 手工配置
  显式地设置要使用的代理。

- 预先配置浏览器
  浏览器厂商或发行商会在将浏览器发送给其客户之前预先对浏览器（或所有其他Web 客户端）的代理设置进行手工配置。 141

- 代理的自动配置（Proxy Auto-Configuration，PAC）
  提供一个 URI，指向一个用 JavaScript 语言编写的代理自动配置文件；客户端会取回这个 JavaScript 文件，并运行它以决定是否应该使用一个代理，如果是的话，应该使用哪个代理服务器。

- WPAD 的代理发现
  有些浏览器支持 Web 代理自动发现协议（Web Proxy Autodiscovery Protocol，WPAD），这个协议会自动检测出浏览器可以从哪个“配置服务器”下载到一个自动配置文件。[7]

6.4.1 客户端的代理配置：手工配置

很多 Web 客户端都允许用户手工配置代理。网景的 Navigator 和微软的 Internet Explorer 都为代理配置提供了便捷的支持。

在网景的 Navigator 6 中，可以通过菜单选项 Edit（编辑）→ Preferences（首选项）→ Advanced（高级）→ Proxies（代理），然后选中单选按钮“Manual proxy”（手工配置代理）来指定代理。

在微软的 Internet Explorer 5 中，可以在 Tools（工具）→ Internet Options（Internet 选项）菜单中，选择一个连接，点击“Settings”（设置），选中“Use a proxy server”（使用代理服务器）框，并点击“Advanced”（高级），来手工指定代理。

其他浏览器都有不同的方式来进行手工配置的修改，但其思想是一样的：为代理指定主机和端口。有些 ISP 会向客户发送预先配置好的浏览器，或定制好的操作系统，使其将 Web 流量重定向到代理服务器上。

6.4.2 客户端代理配置：PAC文件

手工代理配置很简单但有些死板。只能为所有内容指定唯一的一个代理服务器，而且不支持故障转移。手工代理配置还会给大型组织带来管理问题。如果配置过的浏

注 7：当前只有 Internet Explorer 支持这一特性。

览器基数很大，那么需要进行修改的时候，重新配置每个浏览器是非常困难，甚至是不可能的。

PAC 文件是一些小型的 JavaScript 程序，可以在运行过程中计算代理设置，因此，是一种更动态的代理配置解决方案。访问每个文档时，JavaScript 函数都会选择恰当的代理服务器。

要使用 PAC 文件，就要用 JavaScript PAC 文件的 URI 来配置浏览器［配置方式与
142 手工配置类似，但要在“automatic configuration”（自动配置）框中提供一个 URI］。浏览器会从这个 URI 上获取 PAC 文件，并用 JavaScript 逻辑为每次访问计算恰当的代理服务器。PAC 文件的后缀通常是 .pac，MIME 类型通常是 application/x-ns-proxy-autoconfig。

每个 PAC 文件都必须定义一个名为 `FindProxyForURL(url,host)` 的函数，用来计算访问 URI 时使用的适当的代理服务器。函数的返回值可以是表 6-1 列出的任意值。

表6-1　代理自动配置脚本的返回值

FindProxyForURL的返回值	描　　述
DIRECT	不经过任何代理，直接进行连接
PROXY host:port	应该使用指定的代理
SOCKS host:port	应该使用指定的 SOCKS 服务器

例 6-1 中的 PAC 文件为 HTTP 事务处理指定了一个代理，为 FTP 事务处理指定了另一个代理，并为所有其他类型的事务处理使用直连方式。

例 6-1　代理自动配置文件示例

```
function FindProxyForURL(url, host) {
    if (url.substring(0,5) == "http:") {
        return "PROXY http-proxy.mydomain.com:8080";
    } else if (url.substring(0,4) =="ftp:") {
        return "PROXY ftp-proxy.mydomain.com:8080";
    } else {
        return "DIRECT";
    }
}
```

更多有关 PAC 文件的细节，请参见第 20 章。

6.4.3　客户端代理配置：WPAD

另一种浏览器配置机制是 WPAD 协议。WPAD 协议的算法会使用发现机制的逐

级上升策略自动地为浏览器查找合适的 PAC 文件。实现 WPAD 协议的客户端需要：

- 用 WPAD 找到 PAC 的 URI；
- 从指定的 URI 获取 PAC 文件；
- 执行 PAC 文件来判定代理服务器；
- 为请求使用代理服务器。

WPAD 会使用一系列的资源发现技术来判定适当的 PAC 文件。并不是所有组织都能够使用所有的发现技术，所以 WPAD 使用了很多种发现技术。WPAD 会一个接一个地对每种技术进行尝试，直到成功为止。 143

当前的 WPAD 协议规范按顺序定义了下列技术：

- 动态主机配置协议（Dynamic Host Configuration Protocol，DHCP）；
- 服务定位协议（Service Location Protocol，SLP）；
- DNS 知名主机名；
- DNS SRV 记录；
- TXT 记录中的 DNS 服务 URI。

更多信息，请参阅第 20 章。

6.5 与代理请求有关的一些棘手问题

本节对与代理服务器请求有关的一些比较棘手且易被误解的问题进行了解释，其中包括：

- 代理请求中的 URI 和服务器请求中的 URI 有何不同；
- 拦截和反向代理是如何将服务器主机信息隐藏起来的；
- 修改 URI 的规则；
- 代理是怎样影响浏览器的智能 URI 自动完成机制，或主机名扩展特性的。

6.5.1 代理URI与服务器URI的不同

除了一点之外，Web 服务器报文和 Web 代理报文的语法是一样的。客户端向服务器而不是代理发送请求时，HTTP 请求报文中的 URI 会有所不同。

客户端向 Web 服务器发送请求时，请求行中只包含部分 URI（没有方案、主机或端口），如下例所示：

```
GET /index.html HTTP/1.0
User-Agent: SuperBrowser v1.3
```

但当客户端向代理发送请求时，请求行中则包含完整的 URI。例如：

```
GET http://www.marys-antiques.com/index.html HTTP/1.0
User-Agent: SuperBrowser v1.3
```

为什么会有两种不同的请求格式，一种用于代理，另一种用于服务器呢？在原始的 HTTP 设计中，客户端会直接与单个服务器进行对话。不存在虚拟主机，也没有为代理制定什么规则。单个的服务器都知道自己的主机名和端口，所以，为了避免发送冗余信息，客户端只需发送部分 URI 即可，无需发送方案和主机（以及端口）。

代理出现之后，使用部分 URI 就有问题了。代理需要知道目标服务器的名称，这样
144 它们才能建立自己与服务器的连接。基于代理的网关要知道 URI 的方案才能连接到 FTP 资源和其他方案上去。HTTP/1.0 要求代理请求发送完整的 URI，解决了这个问题，但它为服务器请求保留部分 URI 的形式（已经有相当多的服务器都改为支持完整 URI 了）。[8]

因此，我们要将部分 URI 发送给服务器，将完整 URI 发送给代理。在显式地配置客户端代理设置的情况下，客户端就知道要发布哪种类型的请求了。

(1) 没有设置客户端使用代理时，它会发送部分 URI（参见图 6-15a）。
(2) 设置客户端使用代理时，它会发送完整 URI（参见图 6-15b）。

6.5.2　与虚拟主机一样的问题

代理"缺少方案 / 主机 / 端口"的问题与虚拟主机 Web 服务器面临的问题相同。虚拟主机 Web 服务器会在很多 Web 站点间共享同一个物理 Web 服务器。包含部分 URI（比如 /index.html）的请求到达时，虚拟主机 Web 服务器需要知道目的 Web 站点的主机名（更多信息请参见 5.7.1 节和 18.2 节）。

尽管它们出现的问题相似，但解决方法却有所不同：

- 显式的代理要求在请求报文中使用完整 URI 来解决这个问题；
- 虚拟主机 Web 服务器要求使用 `Host` 首部来承载主机和端口信息。

注 8：现在，HTTP/1.1 要求服务器为代理请求和服务器请求都提供完整的 URI 处理，但实际上，很多已部署的服务器仍然只接受部分 URI。

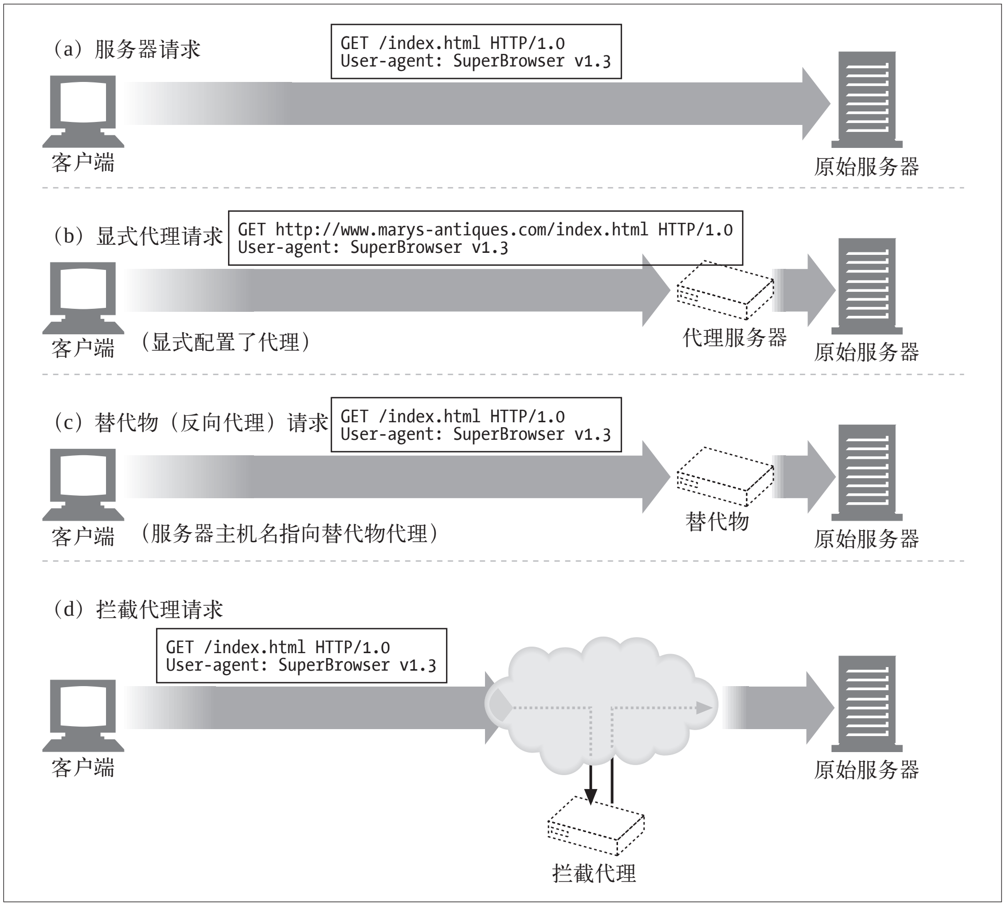


图 6-15　拦截代理会获取服务器请求

145

6.5.3　拦截代理会收到部分URI

只要客户端正确地实现了 HTTP，它们就会在请求中包含完整的 URI，发送给经过显式配置的代理。这样解决了部分问题，但还有一个问题：客户端并不总是知道它是在和代理进行对话，因为有些代理对客户端可能是不可见的。即使没有将客户端配置为使用代理，客户端的流量也可能会经过替代物或拦截代理。在这两种情况下，客户端都会认为它在与 Web 服务器进行对话，不会发送完整的 URI。

- 如前所述，反向代理是一个用来取代原始服务器的代理服务器，它通常会通过假扮服务器的主机名或 IP 地址来做到这一点。它会收到 Web 服务器请求，可能会向真正的服务器提供缓存的响应或者代理请求。客户端无法区分反向代理和 Web 服务器，因此它会发送部分 URI（参见图 6-15c）。

- *拦截代理*是网络流量中的代理服务器，它会拦截从客户端发往服务器的请求，并提供一个缓存响应，或对其进行转发。由于拦截代理拦截了从客户端到服务器的流量，所以它会收到发送给 Web 服务器的部分 URI（参见图 6-15d）。[9]

6.5.4 代理既可以处理代理请求，也可以处理服务器请求

由于将流量重定向到代理服务器的方式有所不同，通用的代理服务器既应该支持请求报文中的完整 URI，也应该支持部分 URI。如果是显式的代理请求，代理就应该
146 使用完整 URI，如果是 Web 服务器请求，就应该使用部分 URI 和虚拟 `Host` 首部。

使用完整和部分 URI 的规则如下所示。

- 如果提供的是完整 URI，代理就应该使用这个完整 URI。
- 如果提供的是部分 URI，而且有 `Host` 首部，就应该用 `Host` 首部来确定原始服务器的名字和端口号。
- 如果提供的是部分 URI，而且没有 `Host` 首部，就要用其他方法来确定原始服务器：
  - 如果代理是代表原始服务器的替代物，可以用真实服务器的地址和端口号来配置代理；
  - 如果流量被拦截了，而且拦截者也可以提供原始的 IP 地址和端口，代理就可以使用拦截技术提供的 IP 地址和端口号（参见第 20 章）；
  - 如果所有方法都失败了，代理没有足够的信息来确定原始服务器，就必须返回一条错误报文（通常是建议用户升级到支持 `Host` 首部的现代浏览器）。[10]

6.5.5 转发过程中对URI的修改

代理服务器要在转发报文时修改请求 URI 的话，需要特别小心。对 URI 的微小修改，甚至是看起来无害的修改，都可能给下游服务器带来一些互操作性问题。

尤其是，现在已知有些代理会在将 URI 转发给下一跳节点之前将 URI“规范”为标准格式。有些看起来无害的转换行为，比如用显式的“:80”来取代默认的 HTTP 端口，或者用适当的换码转义符来取代非法的保留字符以校正 URI，就可能造成互操作性问题。

总之，代理服务器要尽量宽容一些。它们的目标不是成为强制实现严格协议一致性

注 9：在某些情况下，拦截代理可能也会拦截客户端到代理的流量，在这种情况下，拦截代理可能会收到完整 URI，并需要对其进行处理。由于显式代理的通信端口通常与 HTTP 使用的端口有所不同（通常是 8080 而不是 80），而且拦截代理通常只对端口 80 进行拦截，所以这种情况并不会经常发生。

注 10：不应该经常这么做。因为用户会收到之前从未收到过的神秘错误页面。

的“协议警察”，因为这样可能会严重破坏之前能正常工作的服务。

特别是，HTTP 规范禁止一般的拦截代理在转发 URI 时重写其绝对路径部分。唯一的例外是可以用“/”来取代空路径。

6.5.6 URI的客户端自动扩展和主机名解析

根据是否有代理，浏览器对请求 URI 的解析会有所不同。没有代理时，浏览器会获取你输入的 URI，尝试着寻找相应的 IP 地址。如果找到了主机名，浏览器会尝试相应的 IP 地址直到获得成功的连接为止。 147

但是，如果没有找到主机，很多浏览器都会尝试着提供某种主机名自动“扩展”机制，以防用户输入的是主机“简短”的缩写形式（回顾一下 2.3.2 节）。[11]

- 很多浏览器会尝试着加入前缀 www. 和后缀 .com，以防用户只输入了常见 Web 站点名的中间部分（比如，人们可以输入 yahoo 而不是 www.yahoo.com）。
- 有些浏览器甚至会将未解析出来的 URI 传递给第三方站点，这个站点会尝试着校正拼写错误，并给出一些用户可能希望访问的 URI 建议。
- 而且，大多数系统中的 DNS 配置允许用户只输入主机名的前缀，然后 DNS 会自动搜索域名。如果用户位于域名 oreilly.com 的范围之内，并输入了主机名 host7，DNS 会自动尝试将其与 host7.oreilly.com 进行匹配。这并不是完整有效的主机名。

6.5.7 没有代理时URI的解析

图 6-16 显示了一个在没有代理的情况下，浏览器进行主机名自动扩展的例子。在第 (2a) ～ (3c) 步中，浏览器会去查找各种主机名，直到找到一个有效主机名为止。

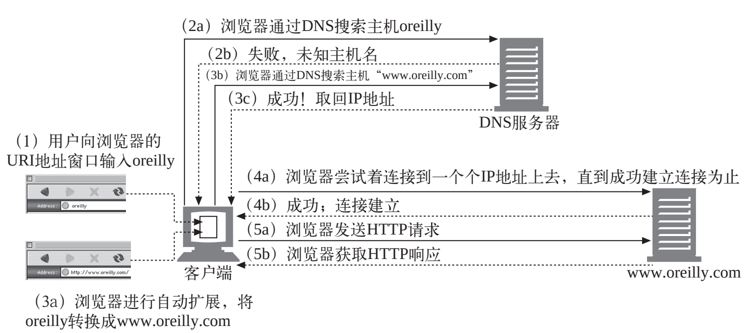


图 6-16　没有提供显式的代理时，浏览器会对部分主机名进行自动扩展

注 11：当你输入 yahoo 时，大多数浏览器都会自动将其扩展为 www.yahoo.com。类似地，浏览器允许用户省略前缀 http://，如果省略了浏览器会自行插入。

图中发生的情况如下所示。

- 在第 (1) 步中，用户向浏览器的 URI 窗口中输入 oreilly。浏览器用 oreilly 作为主机名，并假定默认方案为 http://，默认端口为 80，默认路径为“/”。
- 148 在第 (2a) 步中，浏览器会去查找主机 oreilly。查找失败了。
- 在第 (3a) 步中，浏览器对主机名进行自动扩展，请求 DNS 解析 www.oreilly.com。这次成功了。
- 然后，浏览器成功地连接 www.oreilly.com。

6.5.8 有显式代理时URI的解析

使用显式代理时，用户的 URI 会被直接发送给代理，所以浏览器就不再执行所有这些便捷的扩展功能了。

如图 6-17 所示，有显式代理时，浏览器没有对不完整的主机名进行自动扩展。因此，当用户在浏览器的地址窗口中输入 oreilly 时，发送给代理的就是 http://oreilly/（浏览器添加了默认方案和路径，但主机名和输入的一样）。

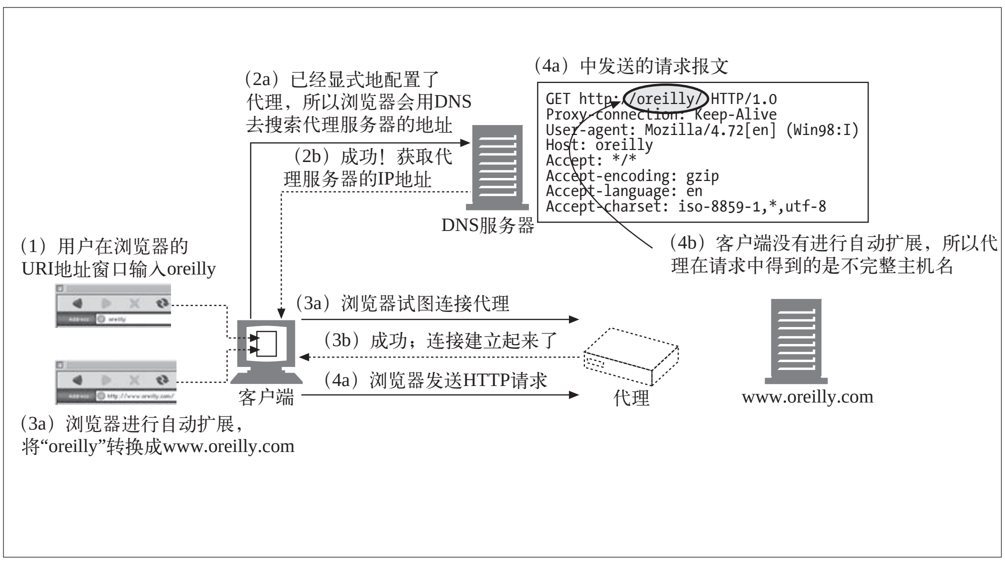


图 6-17 有显式代理时，浏览器不会对不完整主机名进行自动扩展

因此，有些代理会尽力尝试着去模仿浏览器的便捷服务，包括 www...com 自动扩展，以及添加本地域名后缀。[12]

注 12：但对得到广泛共享的代理来说，知道单个用户的正确域名后缀基本上是不可能的。

6.5.9 有拦截代理时URI的解析

使用不可见的拦截代理时，对主机名的解析会略有不同，因为对客户端来说，是没有代理的！这种情况下的行为与使用服务器的情形很类似，浏览器会自动扩展主机名，直到 DNS 成功为止。但如图 6-18 所示，建立到服务器的连接时，有一个很重要的区别。 149

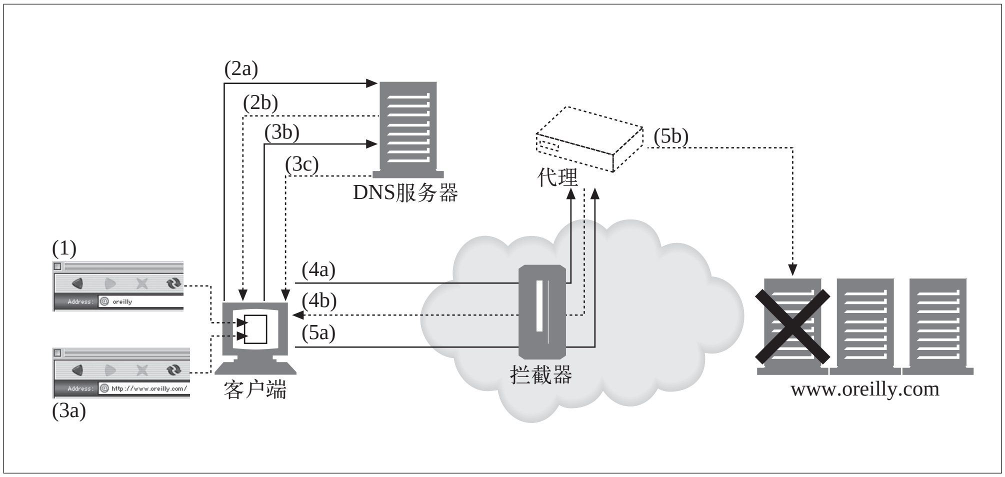


图 6-18　使用拦截代理时，浏览器无法检测出已停用服务器的 IP 地址

图 6-18 显示了下列事务处理过程。

- 在第 (1) 步中，用户在浏览器的 URI 地址窗口中输入 oreilly。
- 在第 (2a) 步中，浏览器通过 DNS 查找主机 oreilly，但 DNS 服务器失败了，并回送响应说明主机未知，如第 (2b) 步所示。
- 在第 (3a) 步中，浏览器进行了自动扩展，将 oreilly 转换成 www.oreilly.com。在第 (3b) 步中，浏览器通过 DNS 来查找主机 www.oreilly.com。这一次，如第 (3c) 步所示，DNS 服务器成功了，将 IP 地址返回给了浏览器。
- 在第 (4a) 步中，客户端已经成功解析了主机名，并有了一张 IP 地址列表。有些 IP 地址可能已经停用了，所以，通常客户端会尝试着连接每个 IP 地址，直到成功为止。但对拦截代理来说，第一次连接请求就会被代理服务器拦截成功，不会连接到原始服务器上去。客户端认为它在与 Web 服务器进行成功的对话，但那个 Web 服务器可能甚至都不处于活跃状态。
- 当代理最终准备好与真正的原始服务器进行交互时［第 (5b) 步］，代理可能会发现那个 IP 地址实际指向的是一个已停用的服务器。为了提供与浏览器相同级别的容错机制，代理可以通过解析 `Host` 首部的主机名，也可以通过对 IP 地址的反

向 DNS 查找来尝试其他 IP 地址。将浏览器配置为使用显式代理时，它们会依赖代理的容错机制，所以对拦截和显式的代理实现来说，在 DNS 解析到已停用服务器时，提供容错机制是很重要的。

6.6　追踪报文

150 现在，在将 Web 请求从客户端传送到服务器的路径上，经过两个或多个代理是很常见的（参见图 6-19）。比如，出于安全和节省费用的考虑，很多公司都会用缓存代理服务器来访问因特网，而且很多大型 ISP 都会使用代理缓存来提高性能并实现各种特性。现在，有相当比例的 Web 请求都是通过代理转发的。同时，出于性能原因，把内容复制到遍布全球的替代物缓存库中的情形也越来越常见了。

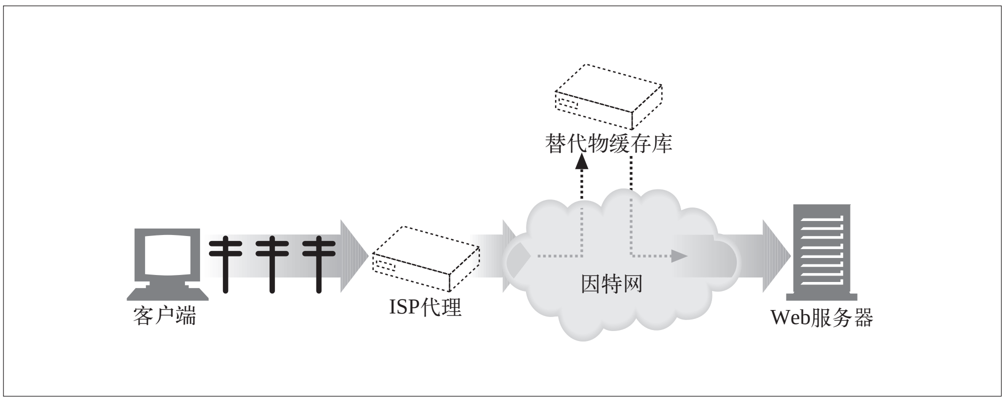


图 6-19　访问代理和 CDN 代理构建了一个两级代理层次结构

代理是由不同厂商开发的。它们有不同的特性和缺陷，由各种不同的组织负责管理。

随着代理的逐渐流行，我们要能够追踪经过代理的报文流，以检测出各种问题，其重要性就跟追踪经过不同交换机和路由器传输的 IP 分组流一样。

6.6.1　`Via`首部

`Via` 首部字段列出了与报文途经的每个中间节点（代理或网关）有关的信息。报文每经过一个节点，都必须将这个中间节点添加到 `Via` 列表的末尾。

下面的 `Via` 字符串告诉我们报文流经了两个代理。这个字符串说明第一个代理名为 `proxy-62.irenes-isp.net`，它实现了 HTTP/1.1 协议，第二个代理被称为 `cache.joes-hardware.com`，实现了 HTTP/1.0：

```
Via: 1.1 proxy-62.irenes-isp.net, 1.0 cache.joes-hardware.com
```

Via 首部字段用于记录报文的转发，诊断报文循环，标识请求 / 响应链上所有发送者的协议能力（参见图 6-20）。

代理也可以用 Via 首部来检测网络中的路由循环。代理应该在发送一条请求之前，在 Via 首部插入一个与其自身有关的独特字符串，并在输入的请求中查找这个字符串，以检测网络中是否存在路由循环。 151

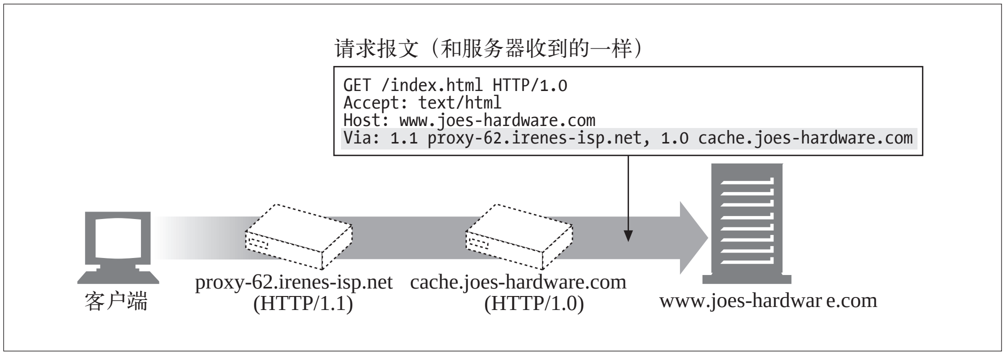


图 6-20　**Via** 首部实例

1. Via的语法

Via 首部字段包含一个由逗号分隔的路标（waypoint）。每个路标都表示一个独立的代理服务器或网关，且包含与那个中间节点的协议和地址有关的信息。下面是一个带有两个路标的 Via 首部实例：

```
Via = 1.1 cache.joes-hardware.com, 1.1 proxy.irenes-isp.net
```

Via 首部的正规语法如下所示：

```
Via               = "Via" ":" 1#( waypoint )
waypoint          = ( received-protocol received-by [ comment ] )
received-protocol = [ protocol-name "/" ] protocol-version
received-by       = ( host [ ":" port ] ) | pseudonym
```

注意，每个 Via 路标中最多包含 4 个组件：一个可选的协议名（默认为 HTTP）、一个必选的协议版本、一个必选的节点名和一个可选的描述性注释。

- *协议名*

  中间节点收到的协议。如果协议为 HTTP 的话，协议名就是可选的。否则，要在版本之前加上协议名，中间用“/”分隔。网关将 HTTP 请求连接到其他协议（HTTPS、FTP 等）时，可能会使用非 HTTP 协议。

- 协议版本

  所收到的报文版本。版本的格式与协议有关。HTTP 使用的是标准版本号（1.0、1.1 等）。版本包含在 `Via` 字段中，这样，之后的应用程序就会知道前面所有中间节点的协议能力了。

- 节点名

  中间节点的主机和可选端口号（如果没有包含端口号，可以假定使用的是协议的默认端口号）。在某些情况下，出于隐私方面的考虑，某个组织可能不愿意给出
152 真实的主机名，在这种情况下可以用一个假名来代替。

- 节点注释

  进一步描述这个中间节点的可选注释。通常会在这里包含厂商和版本信息，有些代理服务器还会在注释字段中包含一些与此设备上所发生事件有关的诊断信息。[13]

2. Via的请求和响应路径

请求和响应报文都会经过代理进行传输，因此，请求和响应报文中都要有 `Via` 首部。

请求和响应通常都是通过同一条 TCP 连接传送的，所以响应报文会沿着与请求报文相同的路径回传。如果一条请求报文经过了代理 A、B 和 C，相应的响应报文就会通过代理 C、B、A 进行传输。因此，响应的 `Via` 首部基本上总是与请求的 `Via` 首部相反（参见图 6-21）。

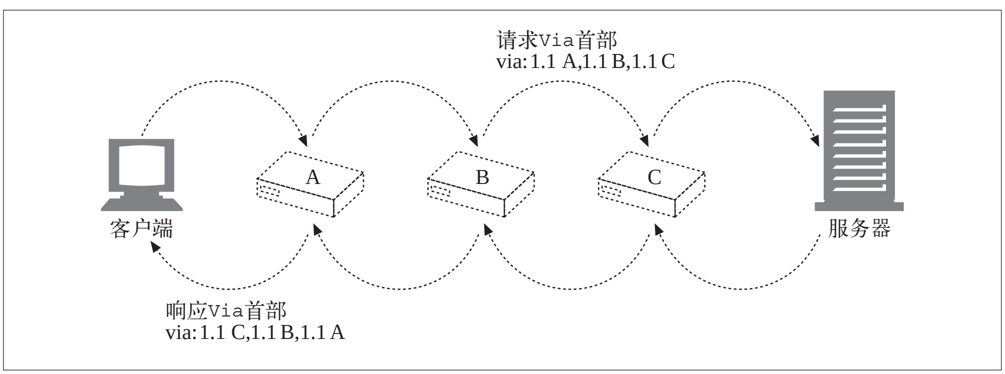


图 6-21　响应 Via 通常与请求 Via 相反

注 13：比如，缓存代理服务器中可能会包含一些成功 / 失败信息。

3. Via与网关

有些代理会为使用非 HTTP 协议的服务器提供网关的功能。Via 首部记录了这些协议转换，这样，HTTP 应用程序就会了解代理链上各点的协议处理能力以及所做的协议转换了。图 6-22 显示了一个通过 HTTP/FTP 网关请求某个 FTP URI 的 HTTP 客户端。

客户端向网关 proxy.irenes-isp.net 发送了一条对 ftp://http-guide.com/pub/welcome.tst 的 HTTP 请求。作为协议网关使用的代理会用 FTP 协议从 FTP 服务器获取预期的对象。然后代理会用下面这个 Via 首部字段，在一条 HTTP 响应中将对象回送到客户端上去：

```
Via: FTP/1.0 proxy.irenes-isp.net (Traffic-Server/5.0.1-17882 [cMs f ])
```

153

注意，接收到的协议是 FTP。可选注释中包含有代理服务器的品牌和版本号，以及一些厂商的诊断信息。在第 8 章可以读到所有与网关有关的内容。

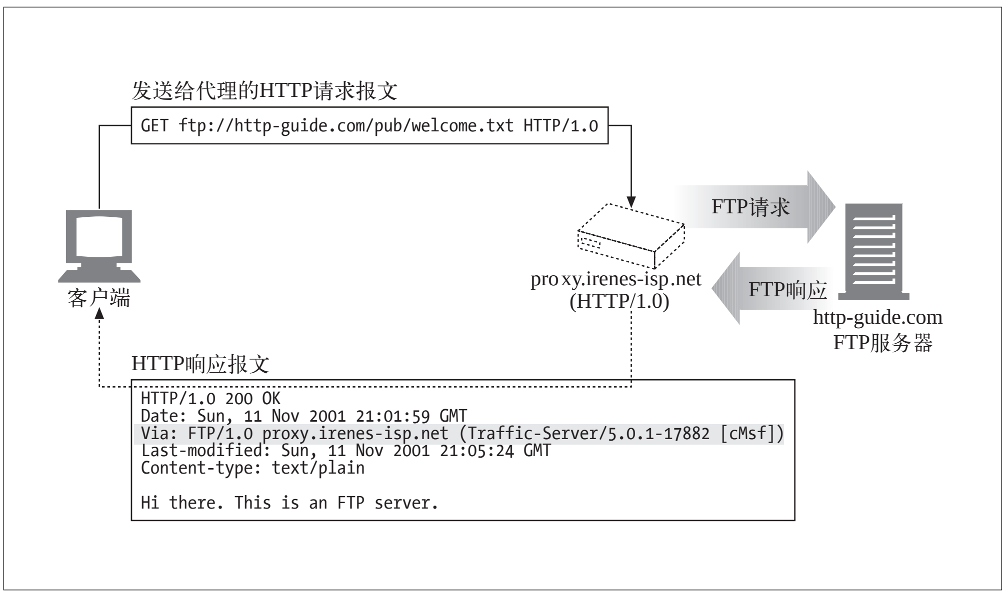


图 6-22　HTTP/FTP 网关生成了 Via 首部，用于记录所收到的协议（FTP）

4. Server和Via首部

Server 响应首部字段对原始服务器使用的软件进行了描述。这里有几个例子：

```
Server: Apache/1.3.14 (Unix) PHP/4.0.4
Server: Netscape-Enterprise/4.1
Server: Microsoft-IIS/5.0
```

如果响应报文是通过代理转发的，一定要确保代理没有修改 Server 首部。Server 首部是用于原始服务器的。代理应该添加的是 Via 条目。

5．Via的隐私和安全问题

有时候，我们并不希望在 Via 字符串中使用确切的主机名。总地来说，除非显式地允许了这种行为，否则，当代理服务器作为网络防火墙的一部分使用时，是不应该转发防火墙后面那些主机的名字和端口号的，因为防火墙后面的网络结构信息可能
154 会被恶意群体利用。[14]

如果不允许进行 Via 节点名转发，作为安全防线的一部分使用的代理就应该用适当的假名来取代那台主机的名字。一般来说，即使隐藏了真实名称，代理也应该尝试着为每台代理服务器保留一个 Via 路标条目。

对那些有着非常强烈的隐私要求，需要隐藏内部网络设计和拓扑结构的组织来说，代理应该将一个（接收协议值相同的）有序 Via 路标条目序列合并成一个联合条目。比如，可以将：

```
Via: 1.0 foo, 1.1 devirus.company.com, 1.1 access-logger.company.com
```

压缩成：

```
Via: 1.0 foo, 1.1 concealed-stuff
```

除非这些条目都在同一个组织的控制之下，而且已经用假名取代了主机名，否则就不能将其合并起来。同样，接收协议值不同的条目也不能合并起来。

6.6.2　TRACE方法

代理服务器可以在转发报文时对其进行修改。可以添加、修改或删除首部，也可以将主体部分转换成不同的格式。代理变得越来越复杂，开发代理产品的厂商也越来越多，互操作性问题也开始逐渐显现。为了便于对代理网络进行诊断，我们需要有一种便捷的方式来观察在通过 HTTP 代理网络逐跳转发报文的过程中，报文是怎样变化的。

通过 HTTP/1.1 的 TRACE 方法，用户可以跟踪经代理链传输的请求报文，观察报文经过了哪些代理，以及每个代理是如何对请求报文进行修改的。TRACE 对代理流的调试非常有用。[15]

注 14：恶意用户可以通过计算机名字和版本号来了解安全防线之后的网络结构。这类信息可能有助于进行安全攻击。而且，计算机名还可能泄露一个组织内部私有项目的线索。

注 15：但是，它还没有得到广泛的实现。

当 TRACE 请求到达目的服务器时，[16] 整条请求报文都会被封装在一条 HTTP 响应的主体中回送给发送端（参见图 6-23）。当 TRACE 响应到达时，客户端可以检查服务器收到的确切报文，以及它所经过的代理列表（在 Via 首部）。TRACE 响应的 Content-Type 为 message/http，状态为 200 OK。

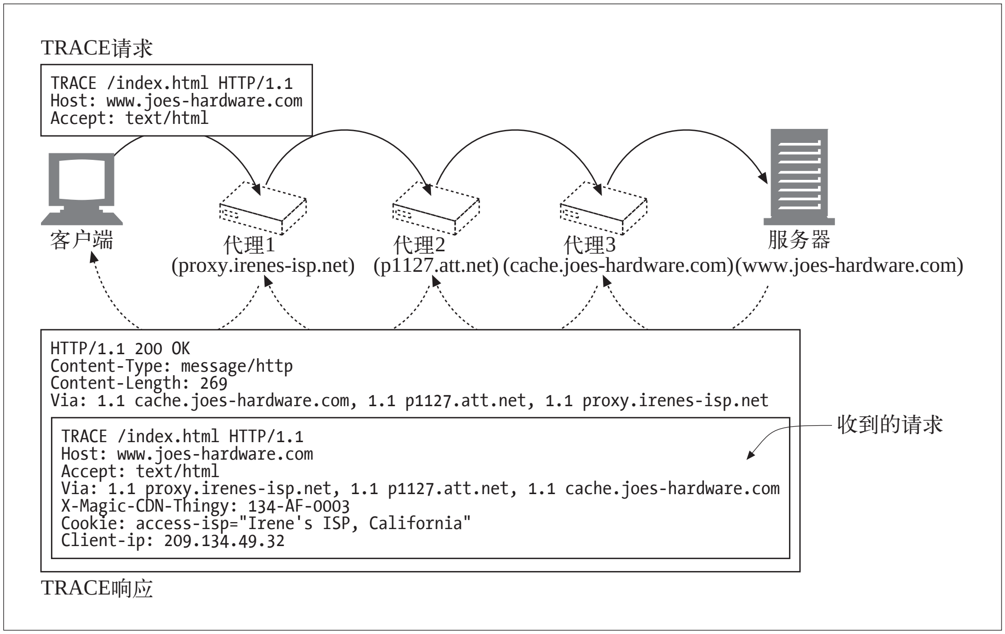


图 6-23　TRACE 响应回送了收到的请求报文

Max-Forwards

通常，不管中间插入了多少代理，TRACE 报文都会沿着整条路径传到目的服务器上。可以使用 Max-Forwards（最大转发次数）首部来限制 TRACE 和 OPTIONS 155
请求所经过的代理跳数，在测试代理链是否是在无限循环中转发报文，或者查看链中特定代理服务器的效果时，它是很有用的。Max-Forwards 也可以限制 OPTIONS 报文的转发（参见 6.8 节）。

Max-Forwards 请求首部字段包含了一个整数，用来说明这条请求报文还可以被转发的次数（参见图 6-24）。如果 Max-Forwards 的值为零（Max-Forwards:0），那么即使接收者不是原始服务器，它也必须将 TRACE 报文回送给客户端，而不应该继续转发。

注 16：最后的接收者可以是原始服务器，也可以是第一个收到了 Max-Forwards 值为零的请求的代理或网关。

如果收到的 Max-Forwards 值大于零，转发的报文中就必须包含一个更新了的 Max-Forwards 字段，其值会被减一。所有的代理和网关都应该支持 Max-Forwards。可以用 Max-Forwards 来查看在代理链的任意一跳上接收到的请求。

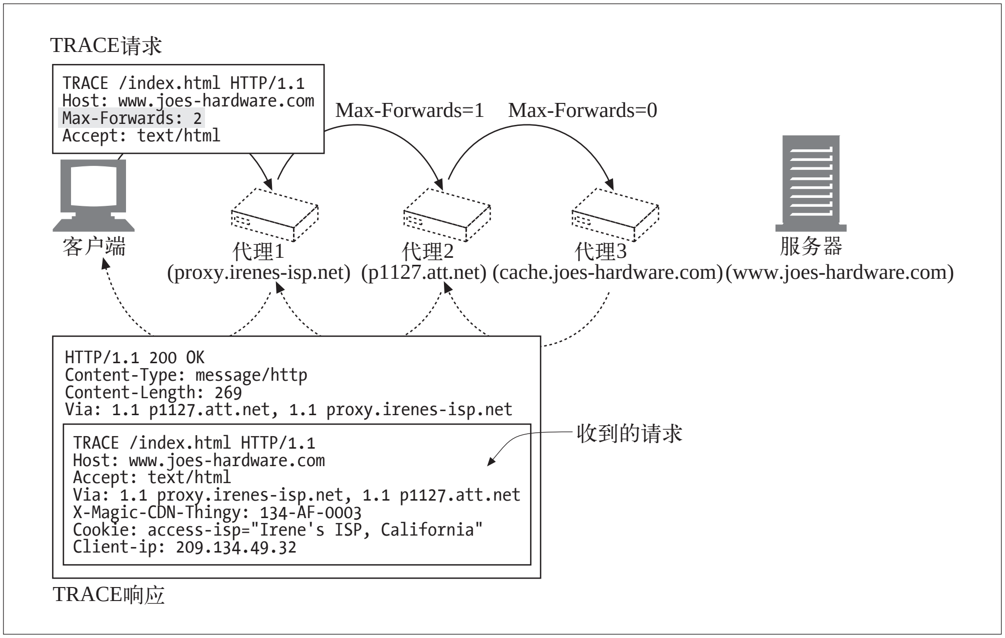


图 6-24　可以用 **Max-Forwards** 首部字段来限制转发跳数

6.7　代理认证

代理可以作为访问控制设备使用。HTTP 定义了一种名为代理认证（proxy authentication）的机制，这种机制可以阻止对内容的请求，直到用户向代理提供了有效的访问权限证书为止。

- 对受限内容的请求到达一台代理服务器时，代理服务器可以返回一个要求使用访问证书的 407 Proxy Authorization Required 状态码，以及一个用于描述怎样提供这些证书的 Proxy-Authenticate 首部字段（参见图 6-25b）。
- 156 客户端收到 407 响应时，会尝试着从本地数据库中，或者通过提示用户来搜集所需要的证书。
- 只要获得了证书，客户端就会重新发送请求，在 Proxy-Authorization 首部字段中提供所要求的证书。
- 如果证书有效，代理就会将原始请求沿着传输链路向下传送（参见图 6-25c）；否则，就发送另一条 407 应答。

若传输链路中有多个代理，且每个代理都要进行认证时，代理认证通常无法很好地工作。人们建议，应该对 HTTP 进行升级，将认证证书与代理链中特定的路标联系起来，但这些升级措施并没有得到广泛实现。

有关 HTTP 认证机制的详细解释请参见第 12 章。

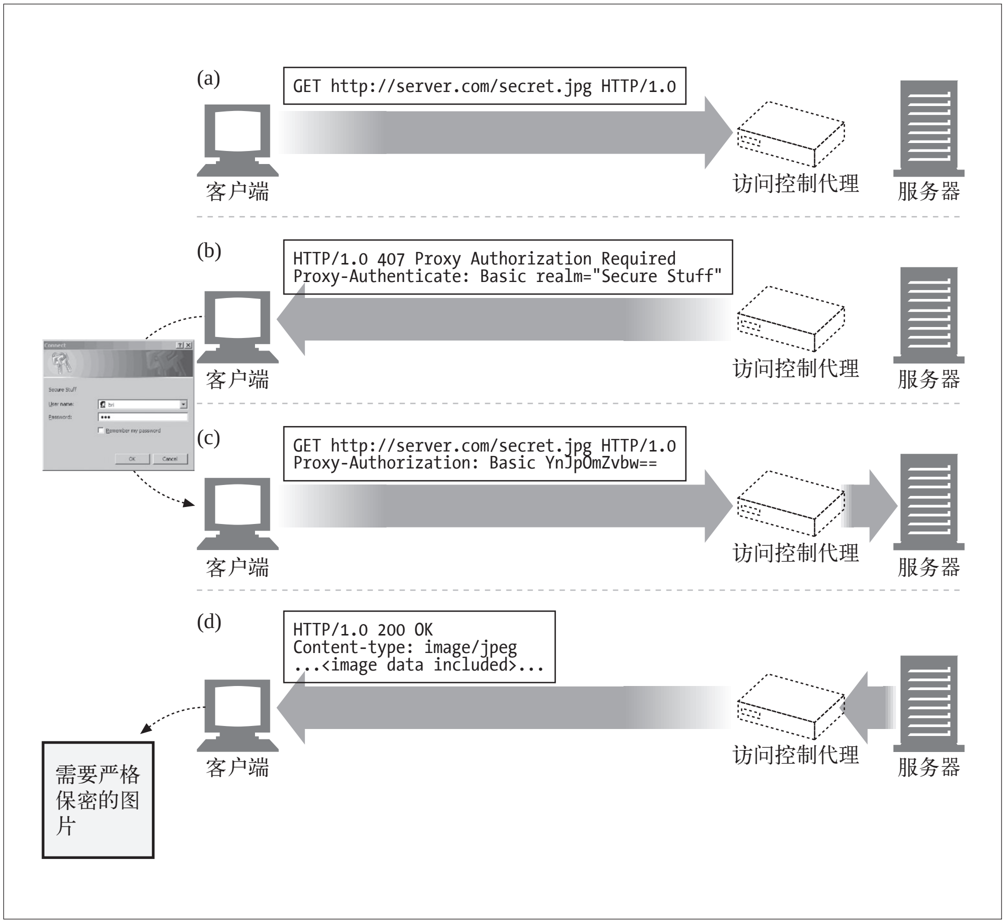


图 6-25　代理可以实现认证机制以控制对内容的访问

6.8　代理的互操作性

客户端、服务器和代理是由不同厂商构建的，实现的是不同版本的 HTTP 规范。它们支持的特性各不相同，也存在着不同的问题。代理服务器位于客户端和服务器设备之间，这些设备实现的协议可能有所不同，可能存在着很棘手的问题。157

6.8.1 处理代理不支持的首部和方法

代理服务器可能无法理解所有经其传输的首部字段。有些首部可能比代理自身还要新；其他首部可能是特定应用程序独有的定制首部。代理必须对不认识的首部字段进行转发，而且必须维持同名首部字段的相对顺序。[17] 类似地，如果代理不熟悉某个方法，那么只要可能，就应该尝试着将报文转发到下一跳节点上去。

在当今的大部分网络中，如果代理不能转发它不支持的方法，可能就无法生存下去

158 了，因为通过微软的 Outlook 进行 Hotmail 访问就大量地使用了 HTTP 扩展方法。

6.8.2 OPTIONS：发现对可选特性的支持

通过 HTTP OPTIONS 方法，客户端（或代理）可以发现 Web 服务器或者其上某个特定资源所支持的功能（比如，它们所支持的方法）（参见图 6-26）。通过使用 OPTIONS，客户端可以在与服务器进行交互之前，确定服务器的能力，这样它就可以更方便地与具备不同特性的代理和服务器进行互操作了。

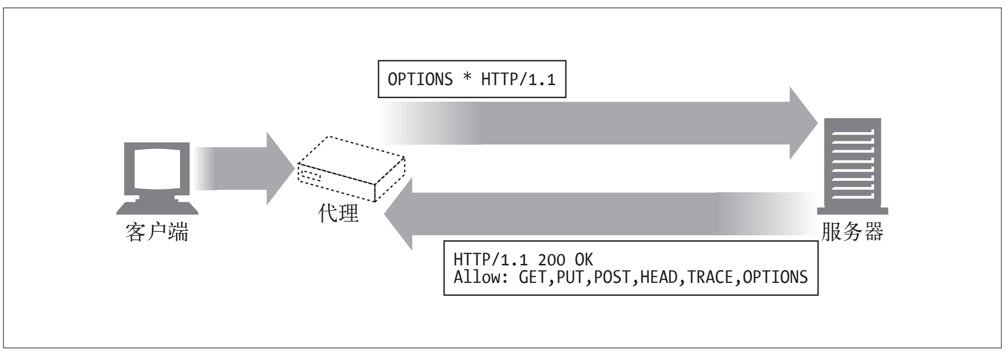


图 6-26 用 OPTIONS 来判定服务器支持的方法

如果 OPTIONS 请求的 URI 是个星号（*），请求的就是整个服务器所支持的功能。比如：

```
OPTIONS * HTTP/1.1
```

如果 URI 是个实际资源地址，OPTIONS 请求就是在查询那个特定资源的可用特性：

```
OPTIONS http://www.joes-hardware.com/index.html HTTP/1.1
```

注 17：报文中可能会出现多个报文首部字段具有相同字段名的情况，如果存在这种情况的话，就要将其等价地合并为一个由逗号分隔的列表。因此，要对合并后的字段值进行解释，具有相同字段名的首部字段的接收顺序就变得非常重要了，因此，代理在转发报文时，就不能修改这些同名字段值的相对顺序。

如果成功，OPTIONS 方法就会返回一个包含了各种首部字段的 200 OK 响应，这些字段描述了服务器所支持的，或资源可用的各种可选特性。HTTP/1.1 在响应中唯一指定的首部字段是 `Allow` 首部，这个首部用于描述服务器所支持的各种方法（或者服务器上的特定资源）。[18]OPTIONS 允许在可选的响应主体中包含更多的信息，但并没有对这种用法进行定义。

6.8.3 `Allow`首部

`Allow` 实体首部字段列出了请求 URI 标识的资源所支持的方法列表，如果请求 URI 为 `*` 的话，列出的就是整个服务器所支持的方法列表。例如：

```
Allow: GET, HEAD, PUT
```

可以将 `Allow` 首部作为请求首部，建议在新的资源上支持某些方法。并不要求服务器支持这些方法，但应该在相应的响应中包含一个 `Allow` 首部，列出它实际支持的方法。 159

因为客户端可能已经通过其他途径与原始服务器进行了交流，所以即使代理无法理解指定的所有方法，也不能对 `Allow` 首部字段进行修改。

6.9 更多信息

更多信息，请参见以下资源。

- http://www.w3.org/Protocols/rfc2616/rfc2616.txt
  RFC 2616，R. Fielding、J. Gettys、J. Mogul、H. Frystyk、L. Mastinter、P. Leach 和 T. Berners-Lee 编写的“Hypertext Transfer Protocol”。
- http://www.ietf.org/rfc/rfc3040.txt
  RFC 3040，“Internet Web Replication and Caching Taxonomy”（“因特网 Web 复制和缓存分类”）。
- *Web Proxy Servers*
  Ari Luotonen 著，Prentice Hall 出版的计算机图书。
- http://www.ietf.org/rfc/rfc3143.txt
  RFC 3143，“Known HTTP Proxy/Caching Problems”（“已知的 HTTP 代理 / 缓存问题”）。

注 18：并不是所有资源都支持每种方法的。比如，CGI 脚本查询可能就不支持文件 PUT，而静态的 HTML 文件则不接受 POST 方法。

- *Web Caching*（《Web 缓存》[19]）
160 Duane Wessels 著，O'Reilly & Associates 公司出版。

注 19：影印版由清华大学出版社出版。（编者注）

第7章

缓　存

Web 缓存是可以自动保存常见文档副本的 HTTP 设备。当 Web 请求抵达缓存时，如果本地有“已缓存的”副本，就可以从本地存储设备而不是原始服务器中提取这个文档。使用缓存有下列优点。

- 缓存减少了冗余的数据传输，节省了你的网络费用。
- 缓存缓解了网络瓶颈的问题。不需要更多的带宽就能够更快地加载页面。
- 缓存降低了对原始服务器的要求。服务器可以更快地响应，避免过载的出现。
- 缓存降低了距离时延，因为从较远的地方加载页面会更慢一些。

本章解释了缓存是怎样提高性能降低费用的、如何去衡量其有效性以及将缓存置于何处可以发挥它的最大作用。我们还会解释 HTTP 如何保持已缓存副本的新鲜度，缓存如何与其他缓存和服务器通信等问题。

7.1 冗余的数据传输

有很多客户端访问一个流行的原始服务器页面时，服务器会多次传输同一份文档，每次传送给一个客户端。一些相同的字节会在网络中一遍遍地传输。这些冗余的数据传输会耗尽昂贵的网络带宽，降低传输速度，加重 Web 服务器的负载。有了缓存，就可以保留第一条服务器响应的副本，后继请求就可以由缓存的副本来应对了，这样可以减少那些流入 / 流出原始服务器的、被浪费掉了的重复流量。

7.2 带宽瓶颈

缓存还可以缓解网络的瓶颈问题。很多网络为本地网络客户端提供的带宽比为远程服务器提供的带宽要宽（参见图 7-1）。客户端会以路径上最慢的网速访问服务器。如果客户端从一个快速局域网的缓存中得到了一份副本，那么缓存就可以提高性
161 能——尤其是要传输比较大的文件时。

在图 7-1 中，Joe 的五金商店旧金山分店的用户通过 1.4Mbit/s 的 T1 因特网连接，从亚特兰大总店下载一个 5MB 的库存文件要花 30 秒的时间。如果在旧金山分店里缓存了这个文档，本地用户通过以太网连接只要花费不到 1 秒的时间就可以获得同一份文档了。

表 7-1 说明了在几种不同的网速下，传输几种不同大小的文档时，带宽会对传输速度产生什么样的影响。带宽会给较大的文档带来显而易见的时延，不同类型网络的速度差异会非常明显。[1] 一个 54kbit/s 的 Modem 传输一个 5MB 的文件需要 749 秒（超过 12 分钟），而在快速以太网 LAN 中，只要不到一秒的时间。

注 1：这张表只列出了网络带宽对传输时间的影响。它假定网络效率为 100%，而且不存在网络或应用程序的处理时延。通过这种方式给出的时延是下限值。实际的时延要大一些，而小型对象的时延则主要是由非带宽开销造成的。

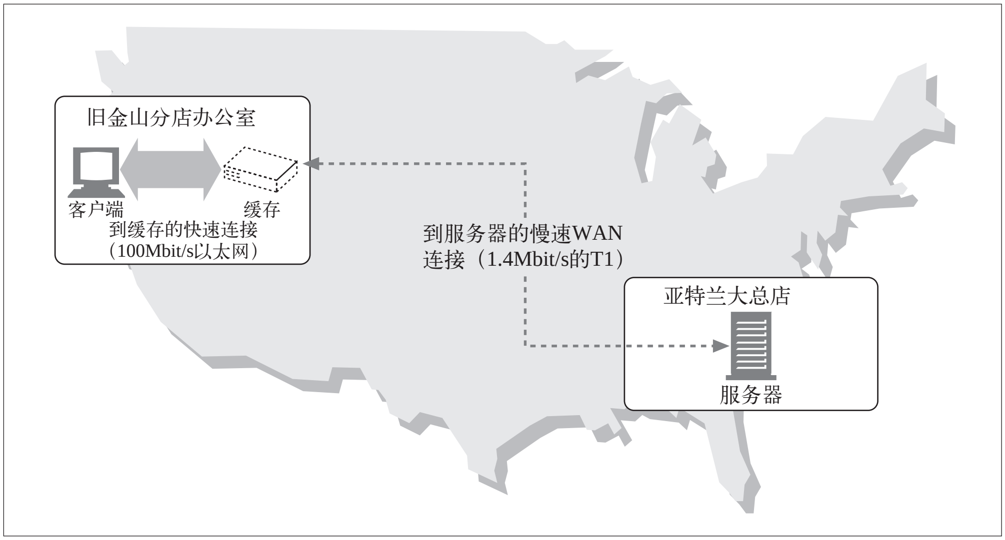


图 7-1　缓存可以改善由有限广域带宽造成的网络瓶颈

表7-1　带宽造成的传输时延，理想化情况（以秒为时间单位）

	大型HTML（15KB）	JPEG（40KB）	大型JPEG（150KB）	大型文件（5MB）
拨号 modem（56kbit/s）	2.19	5.85	21.94	748.98
DSL（256Kbit/s）	0.48	1.28	4.80	163.84
T1（1.4Mbit/s）	0.09	0.23	0.85	29.13
慢速以太网（10Mbit/s）	0.01	0.03	0.12	4.19
DS3（45Mbit/s）	0.00	0.01	0.03	0.93
快速以太网（100Mbit/s）	0.00	0.00	0.01	0.42

162

7.3　瞬间拥塞

缓存在破坏瞬间拥塞（Flash Crowds）时显得非常重要。突发事件（比如爆炸性新闻、批量 E-mail 公告，或者某个名人事件）使很多人几乎同时去访问一个 Web 文档时，就会出现瞬间拥塞（参见图 7-2）。由此造成的过多流量峰值可能会使网络和 Web 服务器产生灾难性的崩溃。

1998 年 9 月 11 日，详细描述 Kenneth Starr 对美国总统克林顿调查情况的“Starr 报告”发布到因特网上去的时候，美国众议院的 Web 服务器每小时收到了超过三百万次的请求，是其平均服务器负荷的 50 倍。据报道，新闻站点 CNN.com 的服务器每秒钟平均会收到超过 50 000 次的请求。

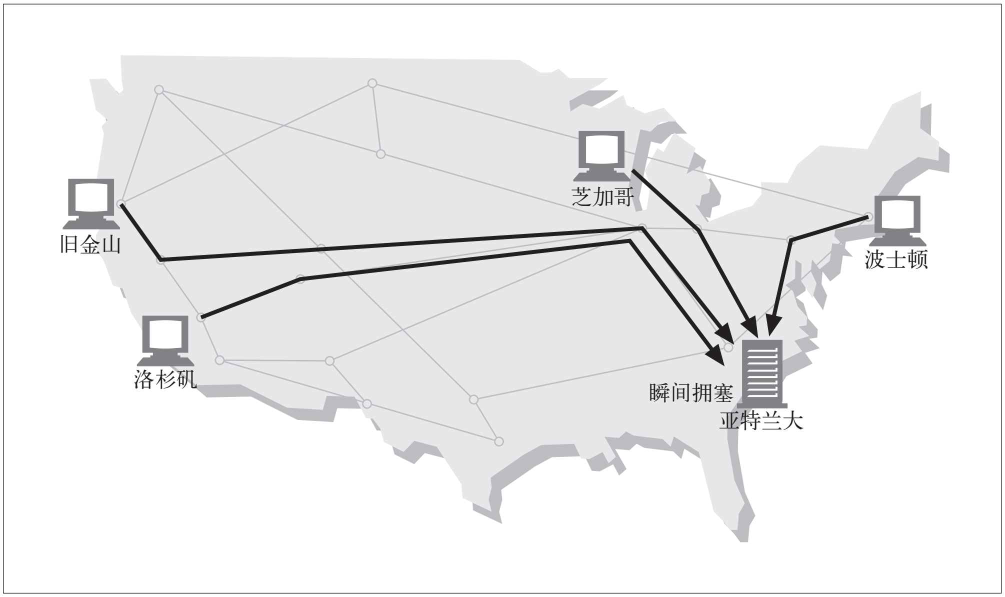


图 7-2　瞬间拥塞会使 Web 服务器过载

7.4　距离时延

即使带宽不是问题，距离也可能成为问题。每台网络路由器都会增加因特网流量的时延。即使客户端和服务器之间没有太多的路由器，光速自身也会造成显著的时延。

波士顿到旧金山的直线距离大约有 2700 英里。在最好的情况下，以光速传输（186 000 英里 / 秒）的信号可以在大约 15 毫秒内从波士顿传送到旧金山，并在 30

163 毫秒内完成一个往返。[2]

假设某个 Web 页面中包含了 20 个小图片，都在旧金山的一台服务器上。如果波士顿的一个客户端打开了 4 条到服务器的并行连接，而且保持着连接的活跃状态，光速自身就要耗费大约 1/4 秒（240 毫秒）的下载时间（参见图 7-3）。如果服务器位于（距离旧金山 6700 英里的）东京，时延就会变成 600 毫秒。中等复杂的 Web 页面会带来几秒钟的光速时延。

将缓存放在附近的机房里可以将文件传输距离从数千英里缩短为数十米。

注 2：在实际应用中，信号的传输速度会比光速低一些，因此，距离时延会更加严重。

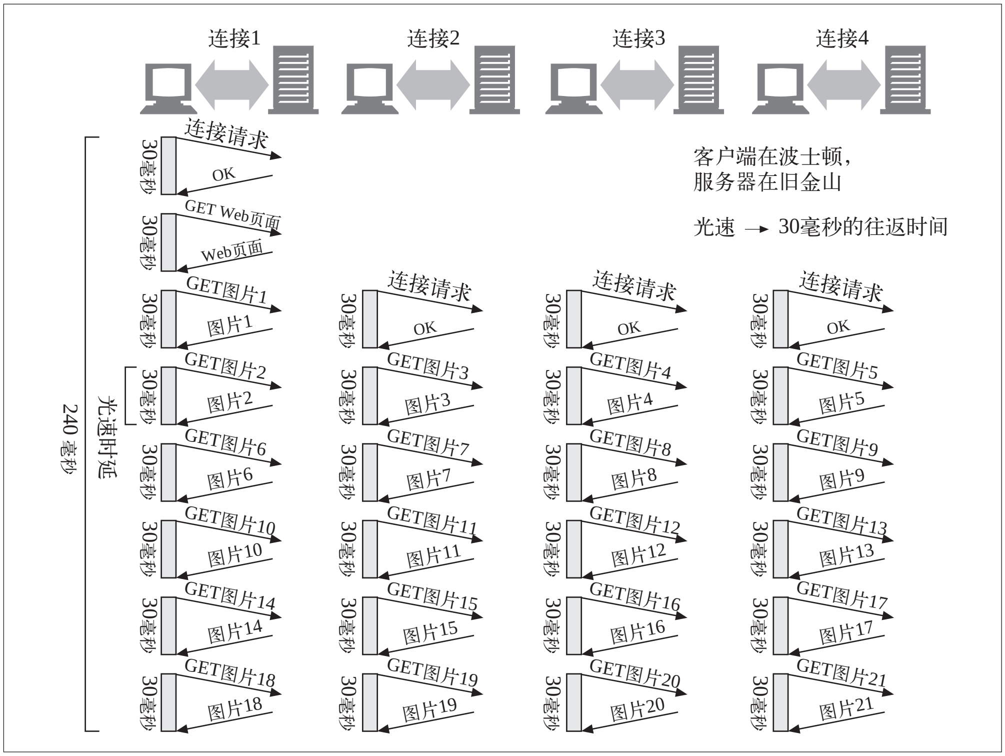


图 7-3　即便使用的是并行的持久连接，光速也会造成显著的时延

7.5　命中和未命中的

这样看来缓存是有所帮助的。但缓存无法保存世界上每份文档的副本。[3] 164

可以用已有的副本为某些到达缓存的请求提供服务。这被称为*缓存命中*（cache hit），参见图 7-4a。其他一些到达缓存的请求可能会由于没有副本可用，而被转发给原始服务器。这被称为*缓存未命中*（cache miss），参见图 7-4b。

7.5.1　再验证

原始服务器的内容可能会发生变化，缓存要不时对其进行检测，看看它们保存的副本是否仍是服务器上最新的副本。这些“新鲜度检测”被称为 HTTP *再验证*（revalidation）（参见图 7-4c）。为了有效地进行再验证，HTTP 定义了一些特殊的请求，不用从服务器上获取整个对象，就可以快速检测出内容是否是最新的。

注 3：几乎没人能够买得起一个大得足以装下 Web 上所有文档的缓存。即便可以买得起巨大的“整个 Web 的缓存”，有些文档也经常会发生变化，很多缓存中的内容都不是最新的。这样的话，在很多缓存中都无法对其进行及时的更新。

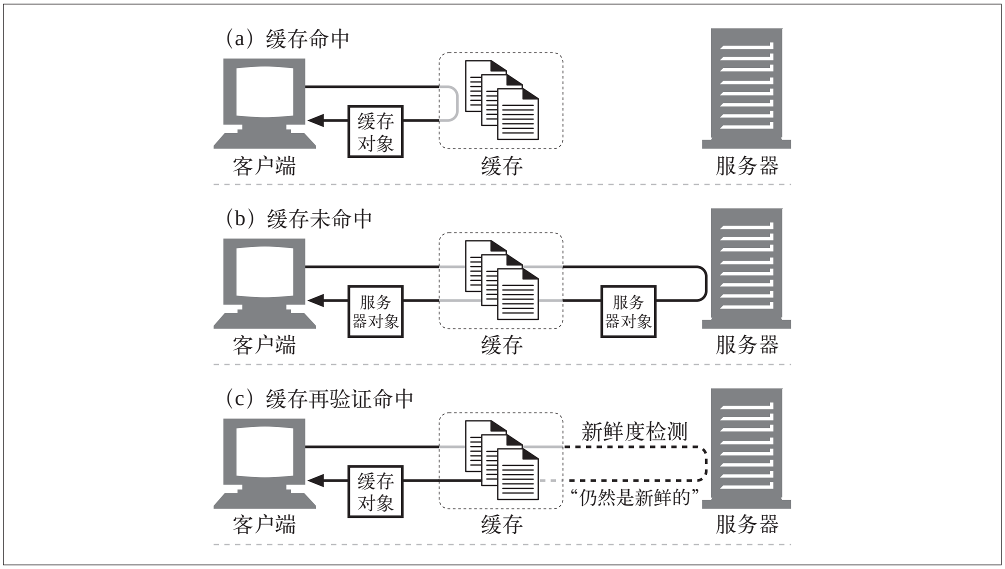


图 7-4　缓存命中、未命中以及再验证

缓存可以在任意时刻，以任意的频率对副本进行再验证。但由于缓存中通常会包含数百万的文档，而且网络带宽是很珍贵的，所以大部分缓存只有在客户端发起请求，并且副本旧得足以需要检测的时候，才会对副本进行再验证。本章稍后会解释 HTTP 的新鲜度检测规则。

缓存对缓存的副本进行再验证时，会向原始服务器发送一个小的再验证请求。如果内容没有变化，服务器会以一个小的 304 Not Modified 进行响应。只要缓存知道副本仍然有效，就会再次将副本标识为暂时新鲜的，并将副本提供给客户端（参见图 7-5a）这被称作*再验证命中*（revalidate hit）或*缓慢命中*（slow hit）。这种方式确实要与原始服务器进行核对，所以会比单纯的缓存命中要慢，但它没有从服务器中获
165 取对象数据，所以要比缓存未命中快一些。

HTTP 为我们提供了几个用来对已缓存对象进行再验证的工具，但最常用的是 `If-Modified-Since` 首部。将这个首部添加到 GET 请求中去，就可以告诉服务器，只有在缓存了对象的副本之后，又对其进行了修改的情况下，才发送此对象。

这里列出了在 3 种情况下（服务器内容未被修改，服务器内容已被修改，或者服务器上的对象被删除了）服务器收到 GET `If-Modified-Since` 请求时会发生的情况：

- *再验证命中*

  如果服务器对象未被修改，服务器会向客户端发送一个小的 HTTP 304 Not Modified 响应。图 7-6 对此进行了描述。

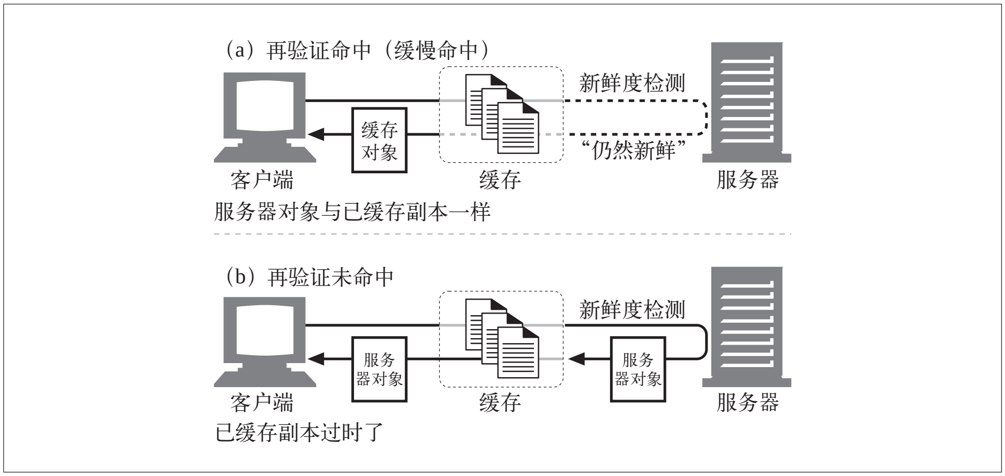


图 7-5　成功的再验证比缓存未命中要快，失败的再验证几乎和未命中的速度一样

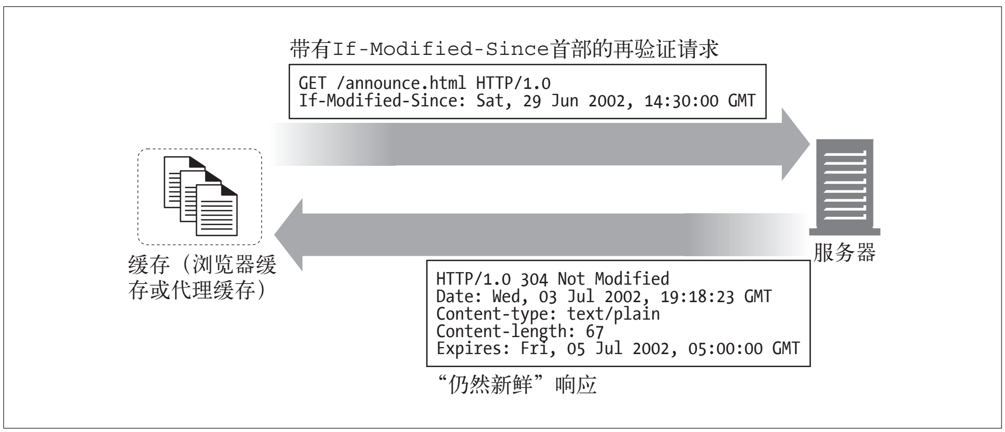


图 7-6　HTTP 使用 **`If-Modified-Since`** 首部进行再验证

- *再验证未命中*

  如果服务器对象与已缓存副本不同，服务器向客户端发送一条普通的、带有完整内容的 HTTP 200 OK 响应。 166

- *对象被删除*

  如果服务器对象已经被删除了，服务器就回送一个 404 Not Found 响应，缓存也会将其副本删除。

7.5.2　命中率

由缓存提供服务的请求所占的比例被称为*缓存命中率*（cache hit rate，或称为缓存

命中比例），[4] 有时也被称为文档命中率（document hit rate）。命中率在 0 到 1 之间，但通常是用百分数来描述的，0% 表示每次请求都未命中（要通过网络来获取文档），100% 表示每次请求都命中了（在缓存中有一份副本）。[5]

缓存的管理者希望缓存命中率接近 100%。而实际得到的命中率则与缓存的大小、缓存用户兴趣点的相似性、缓存数据的变化或个性化频率，以及如何配置缓存有关。命中率很难预测，但对现在中等规模的 Web 缓存来说，40% 的命中率是很合理的。缓存的好处是，即使是中等规模的缓存，其所包含的常见文档也足以显著地提高性能、减少流量了。缓存会努力确保将有用的内容保存在缓存中。

7.5.3　字节命中率

由于文档并不全是同一尺寸的，所以文档命中率并不能说明一切。有些大型对象被访问的次数可能较少，但由于尺寸的原因，对整个数据流量的贡献却更大。因此，有些人更愿意使用字节命中率（byte hit rate）作为度量值（尤其那些按流量字节付费的人！）。

字节命中率表示的是缓存提供的字节在传输的所有字节中所占的比例。通过这种度量方式，可以得知节省流量的程度。100% 的字节命中率说明每个字节都来自缓存，没有流量流到因特网上去。

文档命中率和字节命中率对缓存性能的评估都是很有用的。文档命中率说明阻止了多少通往外部网络的 Web 事务。事务有一个通常都很大的固定时间成分（比如，建立一条到服务器的 TCP 连接），提高文档命中率对降低整体延迟（时延）很有好处。
167 字节命中率说明阻止了多少字节传向因特网。提高字节命中率对节省带宽很有利。

7.5.4　区分命中和未命中的情况

不幸的是，HTTP 没有为用户提供一种手段来区分响应是缓存命中的，还是访问原始服务器得到的。在这两种情况下，响应码都是 200 OK，说明响应有主体部分。有些商业代理缓存会在 `Via` 首部附加一些额外信息，以描述缓存中发生的情况。

客户端有一种方法可以判断响应是否来自缓存，就是使用 `Date` 首部。将响应中 `Date` 首部的值与当前时间进行比较，如果响应中的日期值比较早，客户端通常就可

注 4：术语“命中比例”可能比“命中率”要好，因为“命中率”会让人错误地想到时间因素。但是“命中率”这个词很常用，所以这里我们也使用它。

注 5：有时，人们会在命中率中包括再验证命中，但有时候命中率和再验证命中率是分别测量的。在检测命中率的时候，要确定自己知道什么才是“命中”。

以认为这是一条缓存的响应。客户端也可以通过 Age 首部来检测缓存的响应，通过这个首部可以分辨出这条响应的使用期（参见附录 C 中的 Age 首部）。

7.6 缓存的拓扑结构

缓存可以是单个用户专用的，也可以是数千名用户共享的。专用缓存被称为*私有缓存*（private cache）。私有缓存是个人的缓存，包含了单个用户最常用的页面（参见图 7-7a）。共享的缓存被称为*公有缓存*（public cache）。公有缓存中包含了某个用户团体的常用页面（参见图 7-7b）。

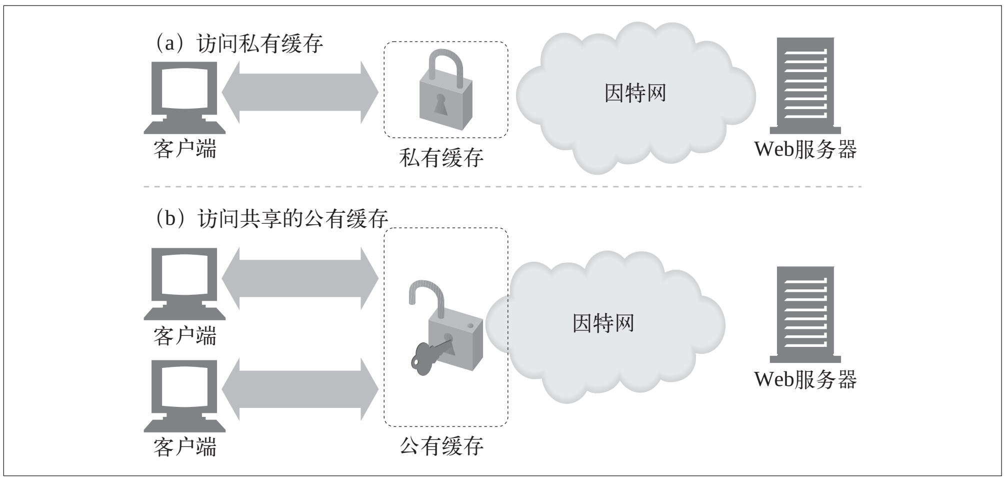


图 7-7　公有和私有缓存

7.6.1 私有缓存

私有缓存不需要很大的动力或存储空间，这样就可以将其做得很小，很便宜。Web 浏览器中有内建的私有缓存——大多数浏览器都会将常用文档缓存在你个人电脑的磁盘和内存中，并且允许用户去配置缓存的大小和各种设置。还可以去看看浏览器的缓存中有些什么内容。比如，对微软的 Internet Explorer 来说，可以从 Tools（工具）→ Internet Options...（因特网选项）对话框中获取缓存内容。MSIE 将缓存的文档称为“临时文件”，并将其与相关的 URL 和文档过期时间一起在文件列表中列出。通过特殊的 URL about:cache 可以查看网景的 Navigator 的缓存内容，这个 URL 会给出一个显示了缓存内容的“磁盘缓存统计”页面。 168

7.6.2 公有代理缓存

公有缓存是特殊的共享代理服务器，被称为*缓存代理服务器*（caching proxy server），

或者更常见地被称为*代理缓存*（proxy cache）（第 6 章讨论过代理）。代理缓存会从本地缓存中提供文档，或者代表用户与服务器进行联系。公有缓存会接受来自多个用户的访问，所以通过它可以更好地减少冗余流量。[6]

在图 7-8a 中，每个客户端都会重复地访问一个（还不在私有缓存中的）新的“热门”文档。每个私有缓存都要获取同一份文档，这样它就会多次穿过网络。而如图 7-8b 所示，使用共享的公有缓存时，对于这个流行的对象，缓存只要取一次就行了，它会用共享的副本为所有的请求服务，以降低网络流量。

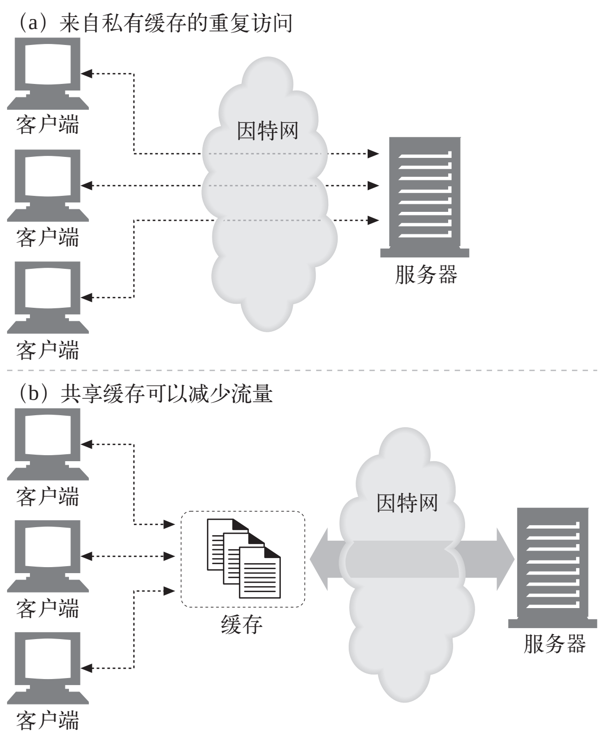


图 7-8　共享的公有缓存可以降低网络流量

代理缓存遵循第 6 章描述的代理规则。可以通过指定手工代理，或者通过代理自动配置文件，将你的浏览器配置为使用代理缓存（参见 6.4.1 节）。还可以通过使用拦截代理在不配置浏览器的情况下，强制 HTTP 请求经过缓存传输（参见第 20 章）。

注 6：公有缓存要缓存用户群体中各种不同的兴趣点，所以要足够大才能承载常用的文档集，而不会被单个用户所感兴趣的文档占满。

7.6.3 代理缓存的层次结构

在实际中，实现*层次化*（hierarchy）的缓存是很有意义的，在这种结构中，在较小缓存中未命中的请求会被导向较大的*父缓存*（parent cache），由它来为剩下的那些“提炼过的”流量提供服务。图 7-9 显示了一个两级的缓存层次结构。[7] 其基本思想是在靠近客户端的地方使用小型廉价缓存，而更高层次中，则逐步采用更大、功能更强的缓存来装载多用户共享的文档。[8]

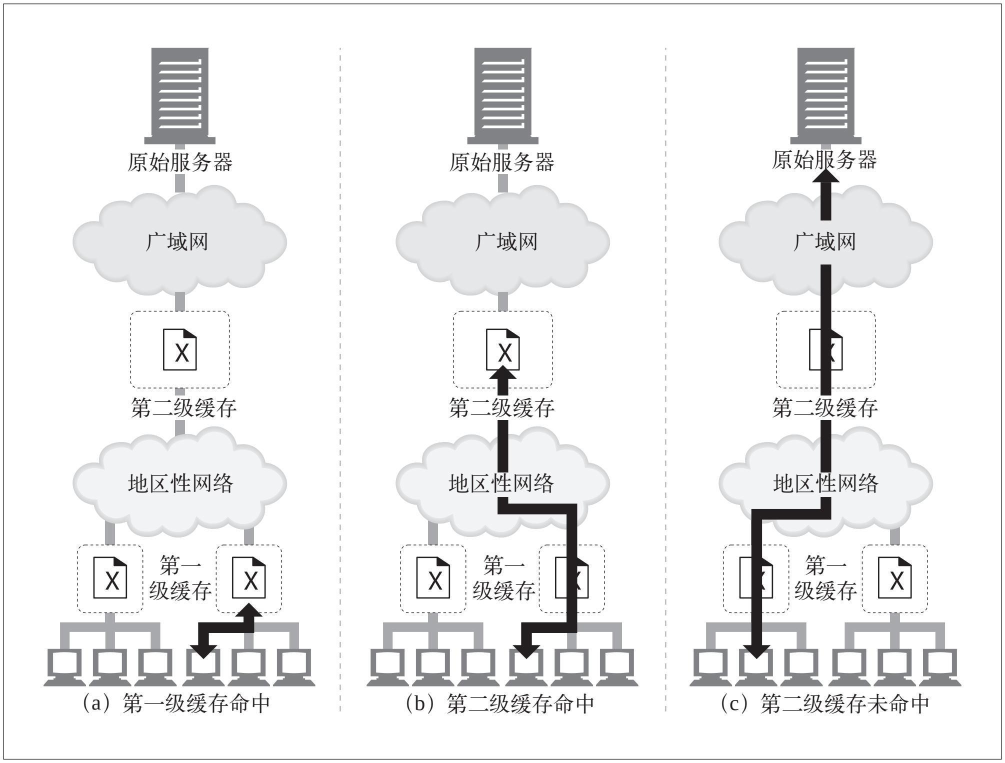


图 7-9 在两级的缓存层次结构中访问文档

我们希望大部分用户都能在附近的第一级缓存中命中（参见图 7-9a）。如果没有命中，较大的父缓存可能能够处理它们的请求（参见图 7-9b）。在缓存层次结构很深 169
的情况下，请求可能要穿过很长一溜缓存，但每个拦截代理都会添加一些性能损耗，当代理链路变得很长的时候，这种性能损耗会变得非常明显。[9]

注 7：如果客户端浏览器自带缓存，那么从技术上来讲，图 7-9 显示的就是一个三级的缓存层次结构。

注 8：父缓存可能要更大一些，以便装载在多用户间流行的文档，它们还要接收来自很多子缓存的聚合流量，这些子缓存的兴趣点可能很分散，所以还需要更高的性能。

注 9：在实际中，网络结构会尝试着将其深度限制在连续的两到三个代理以内。但是，新一代的高性能代理服务器会使代理链的长度变得不那么重要。

7.6.4　网状缓存、内容路由以及对等缓存

有些网络结构会构建复杂的*网状缓存*（cache mesh），而不是简单的缓存层次结构。网状缓存中的代理缓存之间会以更加复杂的方式进行对话，做出动态的缓存通信决策，决定与哪个父缓存进行对话，或者决定彻底绕开缓存，直接连接原始服务器。这种代理缓存会决定选择何种路由对内容进行访问、管理和传送，因此可将其称为*内容路由器*（content router）。

网状缓存中为内容路由设计的缓存（除了其他任务之外）要完成下列所有功能。

- 根据 URL 在父缓存或原始服务器之间进行动态选择。
170 - 根据 URL 动态地选择一个特定的父缓存。
- 前往父缓存之前，在本地缓存中搜索已缓存的副本。
- 允许其他缓存对其缓存的部分内容进行访问，但不允许因特网流量通过它们的缓存。

缓存之间这些更为复杂的关系允许不同的组织互为*对等*（peer）实体，将它们的缓存连接起来以实现共赢。提供可选的对等支持的缓存被称为*兄弟缓存*（sibling cache）（参见图 7-10）。HTTP 并不支持兄弟缓存，所以人们通过一些协议对 HTTP 进行了扩展，比如因特网缓存协议（Internet Cache Protocol，ICP）和超文本缓存协议（HyperText Caching Protocol，HTCP）。我们将在第 20 章讨论这些协议。

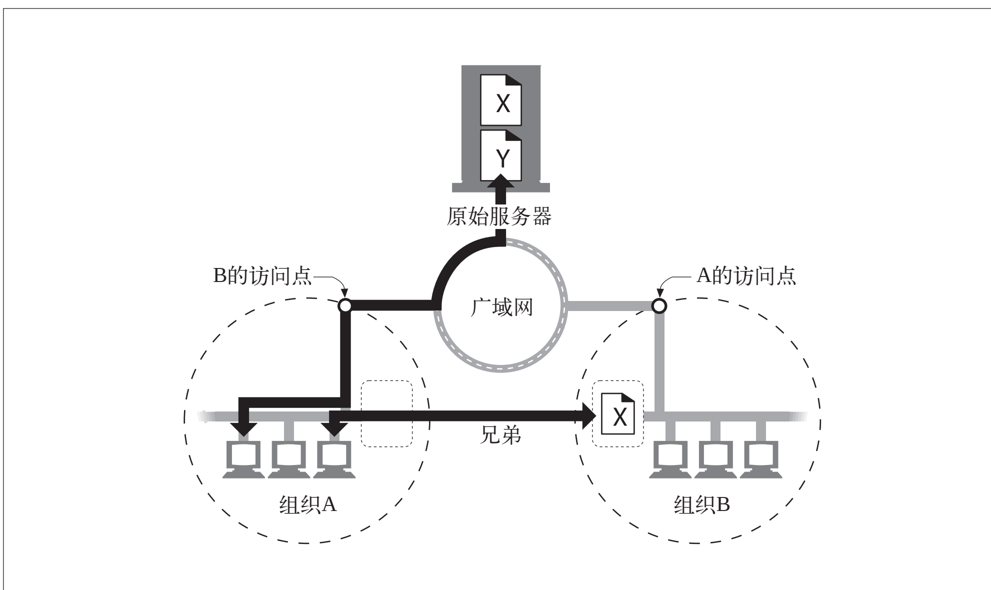


图 7-10　兄弟缓存

7.7 缓存的处理步骤

现代的商业化代理缓存相当地复杂。这些缓存构建得非常高效，可以支持 HTTP 和其他一些技术的各种高级特性。但除了一些微妙的细节之外，Web 缓存的基本工作原理大多很简单。对一条 HTTP GET 报文的基本缓存处理过程包括 7 个步骤（参见图 7-11）。

(1) 接收——缓存从网络中读取抵达的请求报文。

(2) 解析——缓存对报文进行解析，提取出 URL 和各种首部。 171

(3) 查询——缓存查看是否有本地副本可用，如果没有，就获取一份副本（并将其保存在本地）。

(4) 新鲜度检测——缓存查看已缓存副本是否足够新鲜，如果不是，就询问服务器是否有任何更新。

(5) 创建响应——缓存会用新的首部和已缓存的主体来构建一条响应报文。

(6) 发送——缓存通过网络将响应发回给客户端。

(7) 日志——缓存可选地创建一个日志文件条目来描述这个事务。

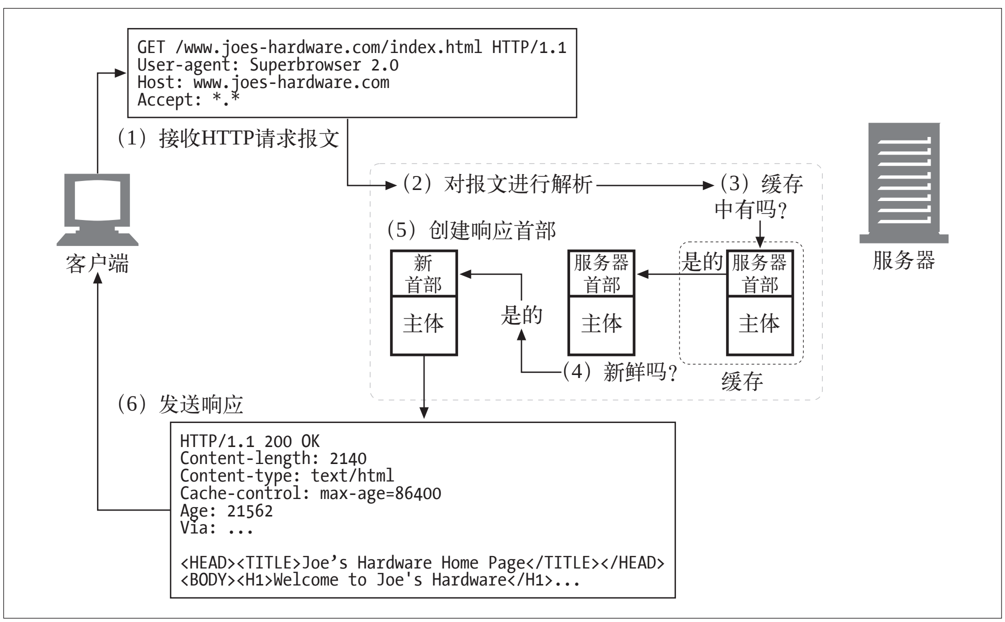


图 7-11　处理一个新鲜的缓存命中

7.7.1 第一步——接收

在第一步中，缓存检测到一条网络连接上的活动，读取输入数据。高性能的缓存会

同时从多条输入连接上读取数据，在整条报文抵达之前开始对事务进行处理。

7.7.2　第二步——解析

接下来，缓存将请求报文解析为片断，将首部的各个部分放入易于操作的数据结构
172 中。这样，缓存软件就更容易处理首部字段并修改它们了。[10]

7.7.3　第三步——查找

在第三步中，缓存获取了 URL，查找本地副本。本地副本可能存储在内存、本地磁盘，甚至附近的另一台计算机中。专业级的缓存会使用快速算法来确定本地缓存中是否有某个对象。如果本地没有这个文档，它可以根据情形和配置，到原始服务器或父代理中去取，或者返回一条错误信息。

已缓存对象中包含了服务器响应主体和原始服务器响应首部，这样就会在缓存命中时返回正确的服务器首部。已缓存对象中还包含了一些*元数据*（metadata），用来记录对象在缓存中停留了多长时间，以及它被用过多少次等。[11]

7.7.4　第四步——新鲜度检测

HTTP 通过缓存将服务器文档的副本保留一段时间。在这段时间里，都认为文档是
173 “新鲜的”，缓存可以在不联系服务器的情况下，直接提供该文档。但一旦已缓存副本停留的时间太长，超过了文档的*新鲜度限值*（freshness limit），就认为对象“过时”了，在提供该文档之前，缓存要再次与服务器进行确认，以查看文档是否发生了变化。客户端发送给缓存的所有请求首部自身都可以强制缓存进行再验证，或者完全避免验证，这使得事情变得更加复杂了。

HTTP 有一组非常复杂的新鲜度检测规则，缓存产品支持的大量配置选项，以及与非 HTTP 新鲜度标准进行互通的需要则使问题变得更加严重了。本章其余的大部分篇幅都用于解释新鲜度的计算问题。

7.7.5　第五步——创建响应

我们希望缓存的响应看起来就像来自原始服务器的一样，缓存将已缓存的服务器响

注 10：解析程序还要负责首部各部分的标准化，将大小写或可替换数据格式之类不太重要的区别都看作等效的。而且，某些请求报文中包含有完整的绝对 URL，而其他一些请求中包含的则是相对 URL 和 `Host` 首部，所以解析程序通常都要将这些细节隐藏起来（参见 2.3.1 节）。

注 11：复杂的缓存还会保留引发服务器响应的原始客户端响应首部的一份副本，以用于 HTTP/1.1 内容协商（参见第 17 章）。

应首部作为响应首部的起点。然后缓存对这些基础首部进行了修改和扩充。

缓存负责对这些首部进行改造，以便与客户端的要求相匹配。比如，服务器返回的可能是一条 HTTP/1.0 响应（甚至是 HTTP/0.9 响应），而客户端期待的是一条 HTTP/1.1 响应，在这种情况下，缓存必须对首部进行相应的转换。缓存还会向其中插入新鲜度信息（Cache-Control、Age 以及 Expires 首部），而且通常会包含一个 Via 首部来说明请求是由一个代理缓存提供的。

注意，缓存不应该调整 Date 首部。Date 首部表示的是原始服务器最初产生这个对象的日期。

7.7.6　第六步——发送

一旦响应首部准备好了，缓存就将响应回送给客户端。和所有代理服务器一样，代理缓存要管理与客户端之间的连接。高性能的缓存会尽力高效地发送数据，通常可以避免在本地缓存和网络 I/O 缓冲区之间进行文档内容的复制。

7.7.7　第七步——日志

大多数缓存都会保存日志文件以及与缓存的使用有关的一些统计数据。每个缓存事务结束之后，缓存都会更新缓存命中和未命中数目的统计数据（以及其他相关的度量值），并将条目插入一个用来显示请求类型、URL 和所发生事件的日志文件。

最常见的缓存日志格式为 Squid 日志格式和网景的可扩展通用日志格式，但很多缓存产品都允许用户创建自定义的日志文件。第 21 章探讨了日志文件格式。 174

7.7.8　缓存处理流程图

图 7-12 以简化形式显示了缓存是如何处理请求，以 GET 一个 URL 的。[12]

7.8　保持副本的新鲜

可能不是所有的已缓存副本都与服务器上的文档一致。毕竟，这些文档会随着时间发生变化。报告可能每个月都会变化。在线报纸每天都会发生变化。财经数据可能每过几秒钟就会发生变化。如果缓存提供的总是老的数据，就会变得毫无用处。已缓存数据要与服务器数据保持一致。

HTTP 有一些简单的机制可以在不要求服务器记住有哪些缓存拥有其文档副本的情

注 12：可以通过条件请求在一步里完成图 7-12 列出的资源再验证及获取（参见 7.8.4 节）。

况下，保持已缓存数据与服务器数据之间充分一致。HTTP 将这些简单的机制称为文档过期（document expiration）和服务器再验证（server revalidation）。

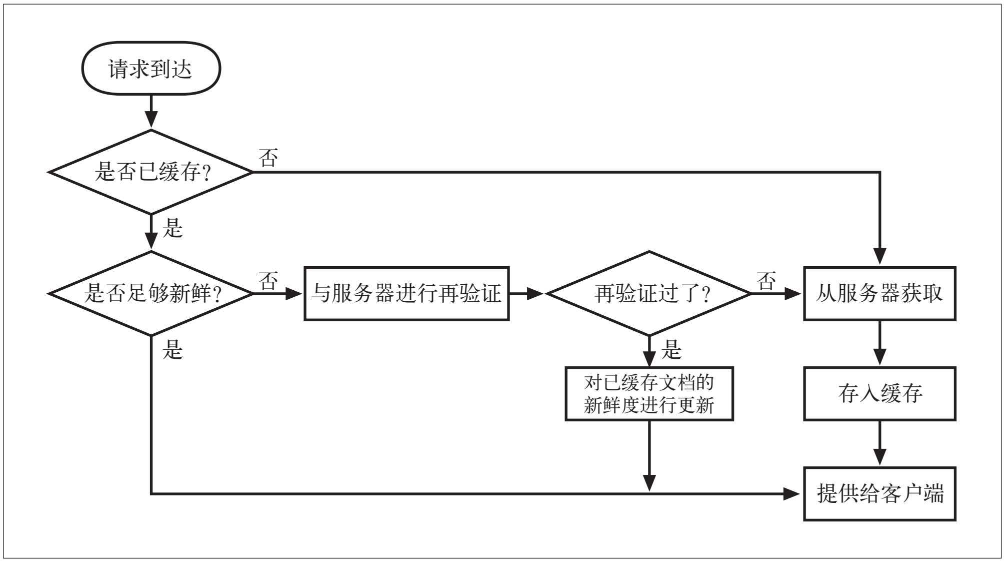


图 7-12　缓存 GET 请求的流程图

7.8.1　文档过期

通过特殊的 HTTP `Cache-Control` 首部和 `Expires` 首部，HTTP 让原始服务器向每个文档附加了一个“过期日期”（参见图 7-13）。就像一夸脱牛奶上的过期日期一
175 样，这些首部说明了在多长时间内可以将这些内容视为新鲜的。

```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Expires: Fri, 05 Jul 2002, 05:00:00 GMT

Independence Day sale at Joe's Hardware
Come shop with us today!
```

（a）`Expires`首部

```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Cache-Control: max-age=484200

Independence Day sale at Joe's Hardware
Come shop with us today!
```

（b）`Cache-Control：max-age`首部

图 7-13　**`Expires`** 首部和 **`Cache-Control`** 首部

在缓存文档过期之前，缓存可以以任意频率使用这些副本，而无需与服务器联系——当然，除非客户端请求中包含有阻止提供已缓存或未验证资源的首部。但一旦已缓存文档过期，缓存就必须与服务器进行核对，询问文档是否被修改过，如果被修改过，就要获取一份新鲜（带有新的过期日期）的副本。

7.8.2 过期日期和使用期

服务器用 HTTP/1.0+ 的 `Expires` 首部或 HTTP/1.1 的 `Cache-Control：max-age` 响应首部来指定过期日期，同时还会带有响应主体。`Expires` 首部和 `Cache-Control：max-age` 首部所做的事情本质上是一样的，但由于 `Cache-Control` 首部使用的是相对时间而不是绝对日期，所以我们更倾向于使用比较新的 `Cache-Control` 首部。绝对日期依赖于计算机时钟的正确设置。表 7-2 列出了各种过期响应首部。

表7-2 过期响应首部

首　部	描　述
`Cache-Control:max-age`	`max-age` 值定义了文档的最大使用期——从第一次生成文档到文档不再新鲜、无法使用为止，最大的合法生存时间（以秒为单位） `Cache-Control: max-age=484200`
`Expires`	指定一个绝对的过期日期。如果过期日期已经过了，就说明文档不再新鲜了 `Expires: Fri, 05 Jul 2002, 05:00:00 GMT`

假设今天是美国东部标准时间（EST，Eastern Standard Time）2002 年 6 月 29 日上午 9：30，Joe 的五金商店正在准备进行 7 月 4 日（美国国庆日）特卖（只剩 5 天了）。Joe 想在他的 Web 服务器上放置一个特殊的 Web 页面，并将其设置为 2002 年 7 月 5 日晚上的 EST 午夜时间过期。如果 Joe 的服务器使用的是老式的 `Expires` 首部，服务器响应报文（参见图 7-13a）中可能就会包含这个首部：[13]

```
Expires: Fri, 05 Jul 2002, 05:00:00 GMT
```
176

如果 Joe 的服务器使用了较新的 `Cache-Control：max-age` 首部，服务器响应报文（参见图 7-13b）中可能就会包含这个首部：

```
Cache-Control: max-age=484200
```

如果这还不够明确的话，可以这样来看，当前时间，EST 时间 2002 年 6 月 29 日早上 9：30，到售卖结束时间 2002 年 7 月 5 日午夜之间有 484 200 秒。到售卖结束之前还有 134.5 小时（大约 5 天）。每小时有 3600 秒，这样到售卖结束之前还有 484 200 秒。

7.8.3 服务器再验证

仅仅是已缓存文档过期了并不意味着它和原始服务器上目前处于活跃状态的文档有

注 13：所有 HTTP 日期和时间都会在格林尼治标准时间（GMT）过期。GMT 是穿过英国格林尼治的本初子午线（经度为零）上的时间。GMT 比美国东部标准时间早五个小时，因此 EST 的午夜就是 05：00GMT。

实际的区别；这只是意味着到了要进行核对的时间了。这种情况被称为“服务器再验证”，说明缓存需要询问原始服务器文档是否发生了变化。

- 如果再验证显示内容*发生了变化*，缓存会获取一份新的文档副本，并将其存储在旧文档的位置上，然后将文档发送给客户端。
- 如果再验证显示内容*没有发生变化*，缓存只需要获取新的首部，包括一个新的过期日期，并对缓存中的首部进行更新就行了。

这是个很棒的系统。缓存并不一定要为每条请求验证文档的有效性——只有在文档过期时它才需要与服务器进行再验证。这样不会提供陈旧的内容，还可以节省服务器的流量，并拥有更好的用户响应时间。

HTTP 协议要求行为正确的缓存返回下列内容之一：

- “足够新鲜”的已缓存副本；
- 与服务器进行过再验证，确认其仍然新鲜的已缓存副本；
- 如果需要与之进行再验证的原始服务器出故障了，就返回一条错误报文[14]；
- 附有警告信息说明内容可能不正确的已缓存副本。

7.8.4 用条件方法进行再验证

HTTP 的条件方法可以高效地实现再验证。HTTP 允许缓存向原始服务器发送一个“条件 GET”，请求服务器只有在文档与缓存中现有的副本不同时，才回送对象主体。通过这种方式，将新鲜度检测和对象获取结合成了单个条件 GET。向 GET 请求报文中添加一些特殊的条件首部，就可以发起条件 GET。只有条件为真时，Web
177 服务器才会返回对象。

HTTP 定义了 5 个条件请求首部。对缓存再验证来说最有用的 2 个首部是 `If-Modified-Since` 和 `If-None-Match`。[15] 所有的条件首部都以前缀“If-”开头。表 7-3 列出了在缓存再验证中使用的条件请求首部。

注 14：如果原始服务器不可访问，但缓存需要进行再验证，那么缓存就必须返回一条错误或一条用来描述通信故障的警告报文。否则，来自已移除服务器上的页面未来可能会在网络的缓存中存留任意长的时间。

注 15：其他条件首部包括 `If-Unmodified-Since`（在进行部分文件的传输时，获取文件的其余部分之前要确保文件未发生变化，此时这个首部是非常有用的）、`If-Range`（支持对不完整文档的缓存）和 `If-Match`（用于与 Web 服务器打交道时的并发控制）。

表7-3　缓存再验证中使用的两个条件首部

首　部	描　述
If-Modified-Since:<date>	如果从指定日期之后文档被修改过了，就执行请求的方法。可以与 Last-Modified 服务器响应首部配合使用，只有在内容被修改后与已缓存版本有所不同的时候才去获取内容
If-None-Match:<tags>	服务器可以为文档提供特殊的标签（参见 ETag），而不是将其与最近修改日期相匹配，这些标签就像序列号一样。如果已缓存标签与服务器文档中的标签有所不同，If-None-Match 首部就会执行所请求的方法

7.8.5　If-Modified-Since:Date再验证

最常见的缓存再验证首部是 If-Modified-Since。If-Modified-Since 再验证请求通常被称为 IMS 请求。只有自某个日期之后资源发生了变化的时候，IMS 请求才会指示服务器执行请求：

- 如果自指定日期后，文档被修改了，If-Modified-Since 条件就为真，通常 GET 就会成功执行。携带新首部的新文档会被返回给缓存，新首部除了其他信息之外，还包含了一个新的过期日期。
- 如果自指定日期后，文档没被修改过，条件就为假，会向客户端返回一个小的 304 Not Modified 响应报文，为了提高有效性，不会返回文档的主体。[16] 这些首部是放在响应中返回的，但只会返回那些需要在源端更新的首部。比如，Content-Type 首部通常不会被修改，所以通常不需要发送。一般会发送一个新的过期日期。

If-Modified-Since 首部可以与 Last-Modified 服务器响应首部配合工作。原始服务器会将最后的修改日期附加到所提供的文档上去。当缓存要对已缓存文档进行再验证时，就会包含一个 If-Modified-Since 首部，其中携带有最后修改已缓存副本的日期： 178

```
If-Modified-Since: <cached last-modified date>
```

如果在此期间内容被修改了，最后的修改日期就会有所不同，原始服务器就会回送新的文档。否则，服务器会注意到缓存的最后修改日期与服务器文档当前的最后修改日期相符，会返回一个 304 Not Modified 响应。

注 16：如果有一个不认识 If-Modified-Since 首部的老服务器收到了条件请求，它会将其作为一个普通的 GET 解释。在这种情况下，系统仍然能够工作，但由于要对未修改的文档数据进行不必要的传输，所以效率会比较低。

例如，如图 7-14 所示，如果你的缓存在 7 月 3 日对 Joe 的五金商店的 7 月 4 日特卖声明进行再验证，就会收到一条 Not Modified 响应（参见图 7-14a）。但如果你的缓存在 7 月 5 日午夜售卖结束后对文档进行再验证，缓存就会收到一个新文档，因为服务器内容已经发生了变化（参见图 7-14b）。

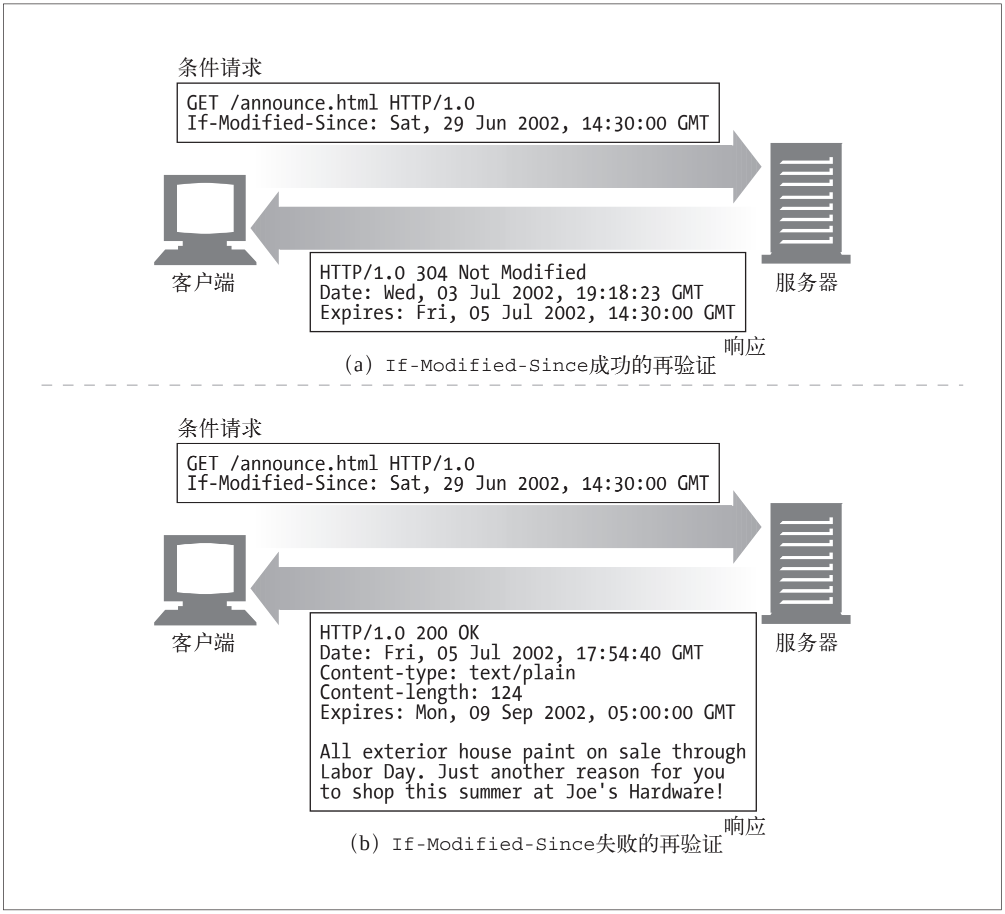


图 7-14　如果未发生变化，`If-Modified-Since` 再验证会返回 304 响应，如果发生了变化，就返回带有新主体的 200 响应

注意，有些 Web 服务器并没有将 `If-Modified-Since` 作为真正的日期来进行比对。相反，它们在 IMS 日期和最后修改日期之间进行了字符串匹配。这样得到的语义就是“如果最后的修改不是在这个确定的日期进行的”，而不是“如果在这个日期
179 之后没有被修改过”。将最后修改日期作为某种序列号使用时，这种替代语义能够很好地识别出缓存是否过期，但这会妨碍客户端将 `If-Modified-Since` 首部用于真正基于时间的一些目的。

7.8.6 If-None-Match：实体标签再验证

有些情况下仅使用最后修改日期进行再验证是不够的。

- 有些文档可能会被周期性地重写（比如，从一个后台进程中写入），但实际包含的数据常常是一样的。尽管内容没有变化，但修改日期会发生变化。
- 有些文档可能被修改了，但所做修改并不重要，不需要让世界范围内的缓存都重装数据（比如对拼写或注释的修改）。
- 有些服务器无法准确地判定其页面的最后修改日期。
- 有些服务器提供的文档会在亚秒间隙发生变化（比如，实时监视器），对这些服务器来说，以一秒为粒度的修改日期可能就不够用了。

为了解决这些问题，HTTP 允许用户对被称为实体标签（ETag）的“版本标识符”进行比较。实体标签是附加到文档上的任意标签（引用字符串）。它们可能包含了文档的序列号或版本名，或者是文档内容的校验和及其他指纹信息。

当发布者对文档进行修改时，可以修改文档的实体标签来说明这个新的版本。这样，如果实体标签被修改了，缓存就可以用 If-None-Match 条件首部来 GET 文档的新副本了。

在图 7-15 中，缓存中有一个实体标签为 v2.6 的文档。它会与原始服务器进行再验证，如果标签 v2.6 不再匹配，就会请求一个新对象。在图 7-15 中，标签仍然与之匹配，因此会返回一条 304 Not Modified 响应。

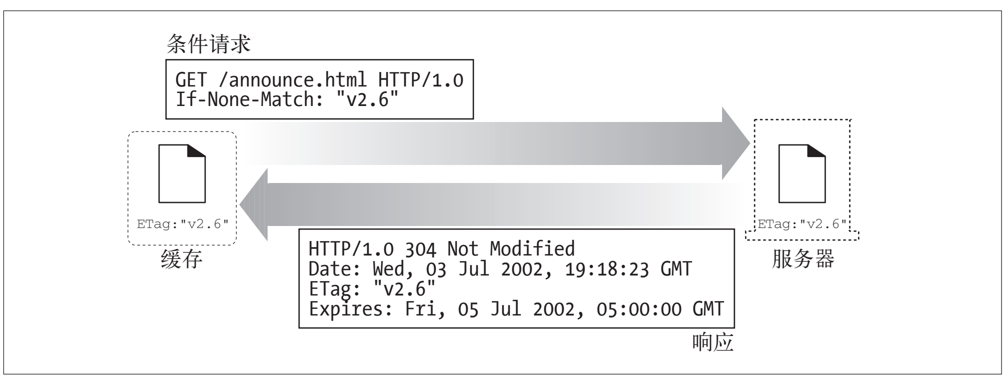


图 7-15 因为实体标签仍然匹配，**If-None-Match** 再验证成功 180

如果服务器上的实体标签已经发生了变化（可能变成了 v3.0），服务器会在一个 200 OK 响应中返回新的内容以及相应的新 Etag。

可以在 If-None-Match 首部包含几个实体标签，告诉服务器，缓存中已经存在带

有这些实体标签的对象副本：

```
If-None-Match: "v2.6"
If-None-Match: "v2.4","v2.5","v2.6"
If-None-Match: "foobar","A34FAC0095","Profiles in Courage"
```

7.8.7　强弱验证器

缓存可以用实体标签来判断，与服务器相比，已缓存版本是不是最新的（与使用最近修改日期的方式很像）。从这个角度来看，实体标签和最近修改日期都是*缓存验证器*（cache validator）。

有时，服务器希望在对文档进行一些非实质性或不重要的修改时，不要使所有的已缓存副本都失效。HTTP/1.1 支持“弱验证器”，如果只对内容进行了少量修改，就允许服务器声明那是“足够好”的等价体。

只要内容发生了变化，强验证器就会变化。弱验证器允许对一些内容进行修改，但内容的主要含义发生变化时，通常它还是会变化的。有些操作不能用弱验证器来实现（比如有条件地获取部分内容），所以，服务器会用前缀“W/”来标识弱验证器。

```
ETag: W/"v2.6"
If-None-Match: W/"v2.6"
```

不管相关的实体值以何种方式发生了变化，强实体标签都要发生变化。而相关实体在语义上发生了比较重要的变化时，弱实体标签也应该发生变化。

注意，原始服务器一定不能为两个不同的实体重用一个特定的强实体标签值，或者为两个语义不同的实体重用一个特定的弱实体标签值。缓存条目都可能会留存任意长的时间，与过期时间无关，有人可能希望当缓存验证条目时，绝对不会再次使用在过去某一时刻获得的验证器，这种愿望可能不太现实。

7.8.8　什么时候应该使用实体标签和最近修改日期

如果服务器回送了一个实体标签，HTTP/1.1 客户端就必须使用实体标签验证器。如果服务器只回送了一个`Last-Modified`值,客户端就可以使用`If-Modified-Since`验证。如果实体标签和最后修改日期都提供了，客户端就应该使用这两种再验证方
181 案，这样 HTTP/1.0 和 HTTP/1.1 缓存就都可以正确响应了。

除非 HTTP/1.1 原始服务器无法生成实体标签验证器，否则就应该发送一个出去，如果使用弱实体标签有优势的话，发送的可能就是个弱实体标签，而不是强实体标签。而且，最好同时发送一个最近修改值。

如果 HTTP/1.1 缓存或服务器收到的请求既带有 If-Modified-Since，又带有实体标签条件首部，那么只有这两个条件都满足时，才能返回 304 Not Modified 响应。

7.9 控制缓存的能力

服务器可以通过 HTTP 定义的几种方式来指定在文档过期之前可以将其缓存多长时间。按照优先级递减的顺序，服务器可以：

- 附加一个 Cache-Control：no-store 首部到响应中去；
- 附加一个 Cache-Control：no-cache 首部到响应中去；
- 附加一个 Cache-Control：must-revalidate 首部到响应中去；
- 附加一个 Cache-Control：max-age 首部到响应中去；
- 附加一个 Expires 日期首部到响应中去；
- 不附加过期信息，让缓存确定自己的过期日期。

本节描述了缓存控制首部。下一节，也就是 7.10 节介绍了如何为不同的内容分配不同的缓存信息。

7.9.1 no-Store与no-Cache响应首部

HTTP/1.1 提供了几种限制对象缓存，或限制提供已缓存对象的方式，以维持对象的新鲜度。no-store 首部和 no-cache 首部可以防止缓存提供未经证实的已缓存对象：

```
Pragma: no-cache
Cache-Control: no-store
Cache-Control: no-cache
```

标识为 no-store 的响应会禁止缓存对响应进行复制。缓存通常会像非缓存代理服务器一样，向客户端转发一条 no-store 响应，然后删除对象。

标识为 no-cache 的响应实际上是可以存储在本地缓存区中的。只是在与原始服务器进行新鲜度再验证之前，缓存不能将其提供给客户端使用。这个首部使用 do-not-serve-from-cache-without-revalidation 这个名字会更恰当一些。

HTTP/1.1 中提供 Pragma：no-cache 首部 [17] 是为了兼容于 HTTP/1.0+。除了与只理解 Pragma：no-cache 的 HTTP/1.0 应用程序进行交互时，HTTP 1.1 应用程序都应该使用 Cache-Control：no-cache。 182

注 17：从技术上来讲，Pragma:no-cache 首部只能用于 HTTP 请求，但在实际中它作为扩展首部已被广泛地用于 HTTP 请求和响应之中。

7.9.2 max-age响应首部

Cache-Control：max-age 首部表示的是从服务器将文档传来之时起，可以认为此文档处于新鲜状态的秒数。还有一个 s-maxage 首部（注意 maxage 的中间没有连字符），其行为与 max-age 类似，但仅适用于共享（公有）缓存：

```
Cache-Control: max-age=3600
Cache-Control: s-maxage=3600
```

服务器可以请求缓存不要缓存文档，或者将最大使用期设置为零，从而在每次访问的时候都进行刷新：

```
Cache-Control: max-age=0
Cache-Control: s-maxage=0
```

7.9.3 Expires响应首部

不推荐使用 Expires 首部，它指定的是实际的过期日期而不是秒数。HTTP 设计者后来认为，由于很多服务器的时钟都不同步，或者不正确，所以最好还是用剩余秒数，而不是绝对时间来表示过期时间。可以通过计算过期值和日期值之间的秒数差来计算类似的新鲜生存期：

```
Expires: Fri, 05 Jul 2002, 05:00:00 GMT
```

有些服务器还会回送一个 Expires:0 响应首部，试图将文档置于永远过期的状态，但这种语法是非法的，可能给某些软件带来问题。应该试着支持这种结构的输入，但不应该产生这种结构的输出。

7.9.4 must-revalidate响应首部

可以配置缓存，使其提供一些陈旧（过期）的对象，以提高性能。如果原始服务器希望缓存严格遵守过期信息，可以在原始响应中附加一个 Cache-Control：must-revalidate 首部。

```
Cache-Control: must-revalidate
```

Cache-Control：must-revalidate 响应首部告诉缓存，在事先没有跟原始服务器进行再验证的情况下，不能提供这个对象的陈旧副本。缓存仍然可以随意提供新鲜的副本。如果在缓存进行 must-revalidate 新鲜度检查时，原始服务器不可
183 用，缓存就必须返回一条 504 Gateway Timeout 错误。

7.9.5 试探性过期

如果响应中没有 `Cache-Control: max-age` 首部，也没有 `Expires` 首部，缓存可以计算出一个试探性最大使用期。可以使用任意算法，但如果得到的最大使用期大于 24 小时，就应该向响应首部添加一个 Heuristic Expiration Warning（试探性过期警告，警告 13）首部。据我们所知，很少有浏览器会为用户提供这种警告信息。

LM-Factor 算法是一种很常用的试探性过期算法，如果文档中包含了最后修改日期，就可以使用这种算法。LM-Factor 算法将最后修改日期作为依据，来估计文档有多么易变。算法的逻辑如下所示。

- 如果已缓存文档最后一次修改发生在很久以前，它可能会是一份稳定的文档，不太会突然发生变化，因此将其继续保存在缓存中会比较安全。
- 如果已缓存文挡最近被修改过，就说明它很可能会频繁地发生变化，因此在与服务器进行再验证之前，只应该将其缓存很短一段时间。

实际的 LM-Factor 算法会计算缓存与服务器对话的时间跟服务器声明文档最后被修改的时间之间的差值，取这个间隔时间的一部分，将其作为缓存中的新鲜度持续时间。下面是 LM-factor 算法的 Perl 伪代码：

```
$time_since_modify = max(0, $server_Date - $server_Last_Modified);
$server_freshness_limit = int($time_since_modify * $lm_factor);
```

图 7-16 以图形方式给出了 LM-factor 的新鲜周期。图中用交叉线画出的阴影表示的是将 LM-factor 设置为 0.2 计算出的新鲜周期。

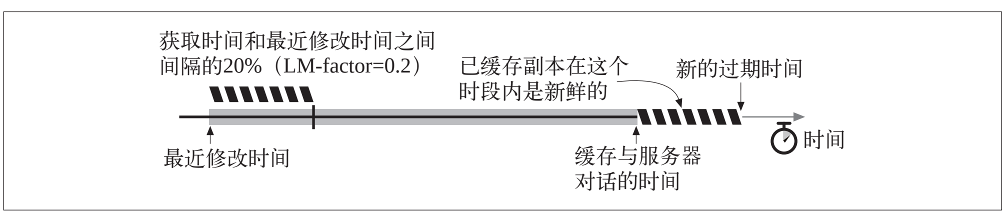


图 7-16　用 LM-factor 算法计算新鲜周期

通常人们会为试探性新鲜周期设置上限，这样它们就不会变得太大了。尽管比较保守的站点会将这个值设置为一天，但通常站点会将其设置为一周。

如果最后修改日期也没有的话，缓存就没什么信息可利用了。缓存通常会为没有任何新鲜周期线索的文档分配一个默认的新鲜周期（通常是一个小时或一天）。有时，比较保守的缓存会将这种试探性新鲜生存期设置为 0，强制缓存在每次将其提供给客户端之前，都去验证一下这些数据仍然是新鲜的。 184

与试探性新鲜计算有关的最后一点是——它们可能比你想象的要常见得多。很多原始服务器仍然不会产生 Expires 和 max-age 首部。选择缓存过期的默认时间时要特别小心！

7.9.6 客户端的新鲜度限制

Web 浏览器都有 Refresh（刷新）或 Reload（重载）按钮，可以强制对浏览器或代理缓存中可能过期的内容进行刷新。Refresh 按钮会发布一个附加了 Cache-Control 请求首部的 GET 请求，这个请求会强制进行再验证，或者无条件地从服务器获取文档。Refresh 的确切行为取决于特定的浏览器、文档以及拦截缓存的配置。

客户端可以用 Cache-Control 请求首部来强化或放松对过期时间的限制。有些应用程序对文档的新鲜度要求很高（比如人工刷新按钮），对这些应用程序来说，客户端可以用 Cache-Control 首部使过期时间更严格。另一方面，作为提高性能、可靠性或开支的一种折衷方式，客户端可能会放松新鲜度要求。表 7-4 对 Cache-Control 请求指令进行了总结。

表7-4 **Cache-Control**请求指令

指　令	目　的
Cache-Control: max-stale Cache-Control: max-stale = <s>	缓存可以随意提供过期的文件。如果指定了参数 <s>，在这段时间内，文档就不能过期。这条指令放松了缓存的规则
Cache-Control: min-fresh=<s>	至少在未来 <s> 秒内文档要保持新鲜。这就使缓存规则更加严格了
Cache-Control: max-age = <s>	缓存无法返回缓存时间长于 <s> 秒的文档。这条指令会使缓存规则更加严格，除非同时还发送了 max-stale 指令，在这种情况下，使用期可能会超过其过期时间
Cache-Control: no-cache Pragma: no-cache	除非资源进行了再验证，否则这个客户端不会接受已缓存的资源
Cache-Control: no-store	缓存应该尽快从存储器中删除文档的所有痕迹，因为其中可能会包含敏感信息
Cache-Control: only-if-cached	只有当缓存中有副本存在时，客户端才会获取一份副本

7.9.7 注意事项

文档过期系统并不是一个完美的系统。如果发布者不小心分配了一个很久之后的过期
185 日期，在文档过期之前，她要对文档做的任何修改都不一定能显示在所有缓存中。[18]

注 18：文档过期采用了“生存时间”技术，这种技术用于很多因特网协议，比如 DNS 中。与 HTTP 一样，如果发布了一个很久之后才到时的过期日期，然后发现需要进行修改，DNS 就会遇到麻烦。但是，与 DNS 不同的是，HTTP 为客户端提供了一些覆盖和强制重载机制。

因此，很多发布者都不会使用很长的过期日期。而且，很多发布者甚至都不使用过期日期，这样缓存就很难确定文档会在多长时间内保持新鲜了。

7.10 设置缓存控制

不同的 Web 服务器为 HTTP `Cache-Control` 和 `Expiration` 首部的设置提供了一些不同的机制。本节简要介绍了流行的 Apache Web 服务器是怎样支持缓存控制的。具体细节请参见你的 Web 服务器文档。

7.10.1 控制Apache的HTTP首部

Apache Web 服务器提供了几种设置 HTTP 缓存控制首部的机制。其中很多机制在默认情况下都没有启动——你要启动它们（有些情况下先要获取 Apache 的扩展模块）。下面是对某些 Apache 特性的简要描述。

- mod_headers

  通过 mod_headers 模块可以对单独的首部进行设置。装载了这个模块，就可以用设置单个 HTTP 首部的指令来扩充 Apache 的配置文件了。还可以将这些设置与 Apache 的常用表达式以及过滤器结合在一起使用，将这些首部与个别内容关联起来。这里有一个配置实例，这个例子将某目录下所有的 HTML 文件都标识为非缓存的：

```
<Files *.html>
  Header set Cache-control no-cache
</Files>
```

- mod_expires

  mod_expires 模块提供的程序逻辑可以自动生成带有正确过期日期的 `Expires` 首部。通过这个模块，就可以将文档的过期日期设置为对其最后一次被访问之后或者其最近修改日期之后的某一时间段。通过这个模块可以为不同的文件类型设置不同的过期日期，还可以使用便捷的详尽描述信息来描述其缓存能力，比如“access plus 1 month（自访问之后起 1 个月）”。这里有几个例子：

```
ExpiresDefault A3600
ExpiresDefault M86400
ExpiresDefault "access plus 1 week"
ExpiresByType text/html "modification plus 2 days 6 hours 12 minutes"
```

- mod_cern_meta

  通过 mod_cern_meta 模块可以将一个包含 HTTP 首部的文件与特定的对象联系起来。启动这个模块时，就创建了一组“元文件”，每个需要控制的文档一个，而且还会为每个元文件添加所期望的首部。 186

7.10.2 通过HTTP-EQUIV控制HTML缓存

HTTP 服务器响应首部用于回送文档的到期信息以及缓存控制信息。Web 服务器与配置文件进行交互，为所提供的文档分配正确的 Cache-Control 首部。

为了让作者在无需与 Web 服务器的配置文件进行交互的情况下，能够更容易地为所提供的 HTML 文档分配 HTTP 首部信息，HTML 2.0 定义了 <META HTTP-EQUIV> 标签。这个可选的标签位于 HTML 文档的顶部，定义了应该与文档有所关联的 HTTP 首部。这里有一个 <META HTTP-EQUIV> 标签设置的例子，它将 HTML 文档标记为非缓冲的：

```
<HTML>
  <HEAD>
    <TITLE>My Document</TITLE>
    <META HTTP-EQUIV="Cache-control" CONTENT="no-cache">
  </HEAD>
       ...
```

最初，HTTP-EQUIV 标签是给 Web 服务器使用的。如 HTML RFC 1866 所述，Web 服务器应该为 HTML 解析 <META HTTP-EQUIV> 标签，并将规定的首部插入 HTTP 响应中：

HTTP 服务器可以用此信息来处理文档。特别是，它可以在为请求此文档的报文所发送的响应中包含一个首部字段：首部名称是从 HTTP-EQUIV 属性值中获取的，首部值是从 CONTENT 属性值中获取的。

不幸的是，支持这个可选特性会增加服务器的额外负载，这些值也只是静态的，而且它只支持 HTML，不支持很多其他的文件类型，所以很少有 Web 服务器和代理支持此特性。

但是，有些浏览器确实会解析并在 HTML 内容中使用 HTTP-EQUIV 标签，像对待真的 HTTP 首部那样来处理嵌入式首部（参见图 7-17）。这样的效果并不好，因为支持 HTTP-EQUIV 标签的 HTML 浏览器使用的 Cache-control 规则可能会与拦截代理缓存所用的规则有所不同。这样会使缓存的过期处理行为发生混乱。

总之，<META HTTP-EQUIV> 标签并不是控制文档缓存特性的好方法。通过配置正确的服务器发出 HTTP 首部，是传送文档缓存控制请求的唯一可靠的方法。

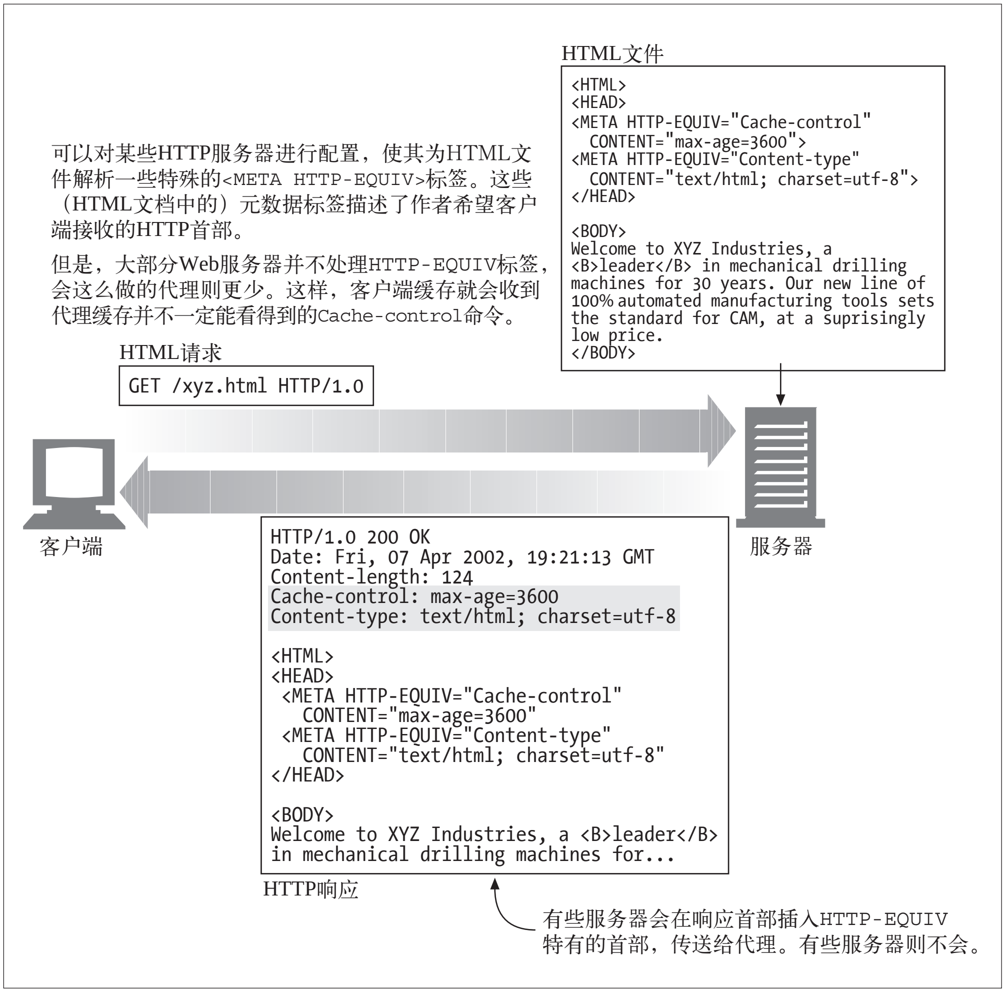


图 7-17　大多数软件都会忽略 **HTTP-EQUIV** 标签，所以这些标签可能会带来一些问题

7.11　详细算法

HTTP 规范提供了一个详细，但有点儿含糊不清而且经常会让人混淆的算法，来计算文档的使用期以及缓存的新鲜度。本节会对 HTTP 的新鲜度计算算法进行详细的讨论（参见图 7-12 中那个“足够新鲜？”菱形框），并对此算法的动机进行解释。 187

本节最适用于那些研究缓存内部机制的人。为了便于说明 HTTP 规范中的内容，我们使用了 Perl 伪代码。如果对计算缓存过期时间的公式中那些繁复的细节不感兴趣的话，可以跳过这一节。 188

7.11.1 使用期和新鲜生存期

为了分辨已缓存文档是否足够新鲜，缓存只需要计算两个值：已缓存副本的*使用期*（age），和已缓存副本的*新鲜生存期*（freshness lifetime）。如果已缓存副本的时长小于新鲜生存期，就说明副本足够新鲜，可以使用。用 Perl 表示为：

```
$is_fresh_enough = ($age < $freshness_lifetime);
```

文档的使用期就是自从服务器将其发送出来（或者最后一次被服务器再验证）之后"老去"的总时间。[19] 缓存可能不知道文档响应是来自上游缓存，还是来自服务器的，所以它不能假设文档是最新的。它必须根据显式的 `Age` 首部（优先），或者通过对服务器生成的 `Date` 首部的处理，来确定文档的使用期。

文档的新鲜生存期表明，已缓存副本在经过多长时间之后，就会因新鲜度不足而无法再向客户端提供了。新鲜生存期考虑了文档的过期日期，以及客户端可能请求的任何新鲜度覆盖范围。

有些客户端可能愿意接受稍微有些过期的文档（使用 `Cache-Control: max-stale` 首部）。有些客户端可能无法接受会在近期过期的文档（使用 `Cache-Control: min-fresh` 首部）。缓存将服务器过期信息与客户端的新鲜度要求结合在一起，以确定最大的新鲜生存期。

7.11.2 使用期的计算

响应的使用期就是服务器发布响应（或服务器对其进行了再验证）之后经过的总时间。使用期包含了响应在因特网路由器和网关中游荡的时间，在中间节点缓存中存储的时间，以及响应在你的缓存中停留的时间。例 7-1 给出了使用期计算的伪代码。

例 7-1 HTTP/1.1 使用期计算算法计算了已缓存文档的总体使用期

```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$corrected_apparent_age = max($apparent_age, $Age_header_value);
$response_delay_estimate = ($time_got_response - $time_issued_request);
$age_when_document_arrived_at_our_cache =
    $corrected_apparent_age + $response_delay_estimate;
$how_long_copy_has_been_in_our_cache = $current_time - $time_got_response;

$age = $age_when_document_arrived_at_our_cache +
    $how_long_copy_has_been_in_our_cache;
```

注 19：记住，服务器上总是有所有文档的最新版本的。

HTTP 使用期计算的细节有点儿棘手，但其基本概念很简单。响应到达缓存时，缓存可以通过查看 Date 首部或 Age 首部来判断响应已使用的时间。缓存还能记录下文档在本地缓存中的停留时间。把这些值加在一起，就是响应的总使用期。HTTP 用一些魔法对时钟偏差和网络时延进行了补偿，但基本计算非常简单： 189

```
$age = $age_when_document_arrived_at_our_cache +
       $how_long_copy_has_been_in_our_cache;
```

缓存可以很方便地判断出已缓存副本已经在本地缓存了多长时间（这就是简单的簿记问题），但很难确定响应抵达缓存时的使用期，因为不是所有服务器的时钟都是同步的，而且我们也不知道响应到过哪里。完善的使用期计算算法会试着对此进行补偿。

1. 表面使用期是基于Date首部的

如果所有的计算机都共享同样的、完全精确的时钟，已缓存文档的使用期就可以是文档的“表面使用期”——当前时间减去服务器发送文档的时间。服务器发送时间就是 Date 首部的值。最简单的起始时间计算可以直接使用表面时间：

```
$apparent_age = $time_got_response - $Date_header_value;
$age_when_document_arrived_at_our_cache = $apparent_age;
```

但并不是所有的时钟都实现了良好的同步。客户端和服务器时钟之间可能有数分钟的差别，如果时钟没有设置好的话，甚至会有数小时或数天的区别。[20]

Web 应用程序，尤其是缓存代理，要做好与时间值有很大差异的服务器进行交互的准备。这种问题被称为*时钟偏差*（clock skew）——两台计算机时钟设置的不同。由于时钟偏差的存在，表面使用期有时会不太准确，而且有时会是负的。

如果使用期是负的，就将其设置为零。我们还可以对表面使用期进行完整性检查，以确定它没有大得令人不可思议，不过，实际上，表面使用期可能并没错。我们可能在与一个将文档缓存了很久的父缓存对话（缓存可能还存储了原始的 Date 首部）：

```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$age_when_document_arrived_at_our_cache = $apparent_age;
```

要明确 Date 首部描述的是原始服务器的日期。代理和缓存一定不能修改这个日期！

注 20：HTTP 规范建议客户端、服务器和代理使用 NTP 这样的时间同步协议来强制使用统一的时间基准。

2. 逐跳使用期的计算

这样就可以去除时钟偏差造成的负数使用期了，但对时钟偏差给精确性带来的整体偏差，我们能做的工作很少。文档经过代理和缓存时，HTTP/1.1 会让每台设备都将相对使用期累加到 Age 首部中去，以此来解决缺乏通用同步时钟的问题。这种方式并不需要进行跨服务器的、端到端的时钟对比。

文档经过代理时，Age 首部值会随之增加。使用 HTTP/1.1 的应用程序应该在 Age
190 首部值中加上文档在每个应用程序和网络传输过程中停留的时间。每个中间应用程序都可以很容易地用本地时钟计算出文档的停留时间。

但响应链中所有的非 HTTP/1.1 设备都无法识别 Age 首部，它们会将首部未经修改地转发出去，或者将其删除掉。因此，在 HTTP/1.1 得到普遍应用之前，Age 首部都将是低估了的相对使用期。

除了基于 Date 计算出来的 Age 之外，还使用了相对 Age 值，而且不论是跨服务器的 Date 值，还是计算出来的 Age 值都可能被低估，所以会选择使用估计出的两个 Age 值中最保守的那个（最保守的值就是最老的 Age 值）。使用这种方式，HTTP 就能容忍 Age 首部存在的错误，尽管这样可能会搞错究竟哪边更新鲜：

```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$corrected_apparent_age = max($apparent_age, $Age_header_value);
$age_when_document_arrived_at_our_cache = $corrected_apparent_age;
```

3. 对网络时延的补偿

事务处理可能会很慢。这是使用缓存的主要动因。但对速度非常慢的网络，或者那些过载的服务器来说，如果文档在网络或服务器中阻塞了很长时间，相对使用期的计算可能会极大地低估文档的使用期。

Date 首部说明了文档是在什么时候离开原始服务器的，[21] 但并没有说明文档在到缓存的传输过程中花费了多长时间。如果文档的传输经过了一长串的代理和父缓存，网络时延可能会相当大。[22]

没有什么简便的方法可以用来测量从服务器到缓存的单向网络时延，但往返时延则比较容易测量。缓存知道它请求文档的时间，以及文档抵达的时间。HTTP/1.1 会在这些网络时延上加上整个往返时延，以便对其进行保守地校正。这个从缓存到服务

注 21：注意，如果文档来自一个父缓存，而不是原始服务器，Date 首部反映的仍是原始服务器，而不是父缓存上的日期。

注 22：实际上，这个时延不会高于几十分之一秒（不然用户就会放弃），但即使是对生存期很短的对象来说，HTTP 的设计者也希望使用尽可能精确的过期时间。

器再到缓存的时延高估了从服务器到缓存的时延，但它是保守的。如果出错了，它只会使文档看起来比实际使用期要老，并引发不必要的再验证。计算是这样进行的： 191

```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$corrected_apparent_age = max($apparent_age, $Age_header_value);
$response_delay_estimate = ($time_got_response - $time_issued_request);
$age_when_document_arrived_at_our_cache =
    $corrected_apparent_age + $response_delay_estimate;
```

7.11.3　完整的使用期计算算法

上一节说明了当 HTTP 所承载的文档抵达缓存时，如何计算其使用期。只要将这条响应存储到缓存中去，它就会进一步老化。当对缓存中文档的请求到达时，我们需要知道文档在缓存中停留了多长的时间，这样才能计算文档现在的使用期：

```
$age = $age_when_document_arrived_at_our_cache +
    $how_long_copy_has_been_in_our_cache;
```

嗒嗒！这样就有了例 7-1 中给出的完整的 HTTP/1.1 使用期计算算法。这就是简单的簿记问题了——我们知道了文档是什么时候到达缓存的（`$time_got_reponse`），也知道当前请求是什么时候到达的（刚才），这样停留时间就是两者之差了。所有这些都以图形方式显示在图 7-18 中了。

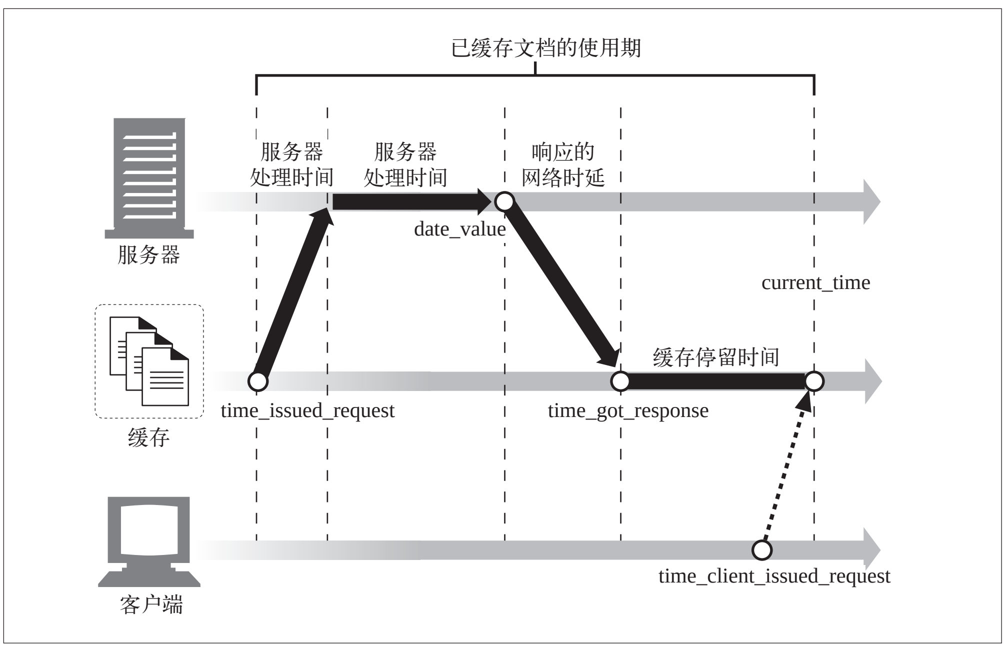


图 7-18　已缓存文档的使用期包括在网络和缓存中停留的时间

7.11.4　新鲜生存期计算

回想一下，我们是在想办法弄清楚已缓存文档是否足够新鲜，是否可以提供给客户端。要回答这个问题，就必须确定已缓存文档的使用期，并根据服务器和客户端限制来计算新鲜生存期。我们刚刚解释了如何计算使用期；现在我们来看看新鲜生存期的计算。

文档的新鲜生存期说明了在文档不再新鲜，无法提供给某个特定的客户端之前能够停
192 留多久。新鲜生存期取决于服务器和客户端的限制。服务器上可能有一些与文档的出版变化率有关的信息。那些非常稳定的已归档报告可能会在数年内保持新鲜。期刊可能只在下一期的出版物出来之前的剩余时间内有效——下一周，或是明早 6 点。

客户端可能有些其他指标。如果稍微有些过期的内容速度更快的话，它们可能也愿意接受，或者它们可能希望接收最新的内容。缓存是为用户服务的。必须要满足他们的要求。

7.11.5　完整的服务器——新鲜度算法

例 7-2 给出了一个用于计算服务器新鲜度限制的 Perl 算法。它会返回文档仍由服务器提供时所能到达的最大使用期。

例 7-2　服务器新鲜度限制的计算

```
sub server_freshness_limit
{
    local($heuristic,$server_freshness_limit,$time_since_last_modify);

    $heuristic = 0;

    if ($Max_Age_value_set)
    {
        $server_freshness_limit = $Max_Age_value;
    }
    elseif ($Expires_value_set)
    {
        $server_freshness_limit = $Expires_value - $Date_value;
    }
    elseif ($Last_Modified_value_set)
    {
        $time_since_last_modify = max(0, $Date_value -
             $Last_Modified_value);
        $server_freshness_limit = int($time_since_last_modify *
             $lm_factor);
        $heuristic = 1;
    }
    else
    {
```

```
        $server_freshness_limit = $default_cache_min_lifetime;
        $heuristic = 1;
    }

    if ($heuristic)
    {
        if ($server_freshness_limit > $default_cache_max_lifetime)
        { $server_freshness_limit = $default_cache_max_lifetime; }
        if ($server_freshness_limit < $default_cache_min_lifetime)
        { $server_freshness_limit = $default_cache_min_lifetime; }
    }

    return($server_freshness_limit);
}
```

193

现在，我们来看看客户端怎样修正服务器为文档指定的使用期限制。例 7-3 显示了一个 Perl 算法，此算法获取了服务器的新鲜度限制并根据客户端的限制对其进行修改。它会返回一个最大使用期，这是在无需再次验证，仍由缓存提供文档的前提下，文档的最大生存时间。

例 7-3　客户端新鲜度限制的计算

```
sub client_modified_freshness_limit
{
    $age_limit = server_freshness_limit( ); ## From Example 7-2

    if ($Max_Stale_value_set)
    {
        if ($Max_Stale_value == $INT_MAX)
        { $age_limit = $INT_MAX; }
        else
        { $age_limit = server_freshness_limit( ) + $Max_Stale_value; }
    }

    if ($Min_Fresh_value_set)
    {
    $age_limit = min($age_limit, server_freshness_limit( ) -
           $Min_Fresh_value_set);
    }

    if ($Max_Age_value_set)
    {
        $age_limit = min($age_limit, $Max_Age_value);
    }
}
```

整个进程中包含两个变量：文档的使用期及其新鲜度限制。如果使用期小于新鲜度限制，就说明文档“足够新鲜”。例 7-3 中的算法只是考虑了服务器的新鲜度限制，并根据附加的客户端限制对其进行了调整。希望通过本节的介绍能使在 HTTP 规范中描述的比较微妙的过期算法更清晰一些。

7.12 缓存和广告

读到这里，你一定已经意识到缓存可以提高性能并减少流量。知道缓存可以帮助用户，并为用户提供更好的使用体验，而且缓存也可以帮助网络运营商减少流量。

7.12.1 发布广告者的两难处境

你可能认为内容提供商会喜欢缓存。毕竟，如果到处都是缓存的话，内容提供商就不需要购买大型的多处理器 Web 服务器来满足用户需求了——他们不需要付过高的
194 网络服务费，一遍一遍地向用户发送同样的数据。更好的一点是，缓存可以将那些漂亮的文章和广告以更快，甚至更好看的方式显示在用户的显示器上，鼓励他们去浏览更多的内容，看更多的广告。这就是内容提供商所希望的！吸引更多的眼球和更多的广告！

但这就是困难所在。很多内容提供商的收益都是通过广告实现的——具体来说，每向用户显示一次广告内容，内容提供商就会得到相应的收益。（可能还不到一两便士，但如果一天显示数百万条广告的话，这些钱就会叠加起来！）这就是缓存的问题——它们会向原始服务器隐藏实际的访问次数。如果缓存工作得很好，原始服务器可能根本收不到任何 HTTP 访问，因为这些访问都被因特网缓存吸收了。但如果你的收益是基于访问次数的话，你就高兴不起来了。

7.12.2 发布者的响应

现在，广告商会使用各种类型的"缓存清除"技术来确保缓存不会窃取他们的命中流量。他们会在内容上加上 `no-cache` 首部。他们会通过 CGI 网关提供广告。还会在每次访问时重写广告 URL。

这些缓存清除技术并不仅用于代理缓存。实际上，现在主要将其用于每个 Web 浏览器中都启用了的缓存。但是，如果某些内容提供商维护其命中率的行为太过火了，就会降低缓存为其站点带来的积极作用。

理想情况下，内容提供商会让缓存吸收其流量，而缓存会告诉内容提供商它们拦截了多少次命中。现在，缓存有好几种方式可以做到这一点。

一种解决方案就是配置缓存，每次访问时都与原始服务器进行再验证。这样，每次访问时都会将命中推向原始服务器，但通常不会传送任何主体数据。当然，这样会降低事务处理的速度。[23]

注 23：有些缓存支持这种再验证的变体形式，在这种方式中，它们可以在后台发起条件 GET 或 HEAD 请求。用户不会感觉到时延，但这个请求会触发对原始服务器的离线访问。这是一种改进方式，但这种方式加重了缓存的负荷，极大地增加了流经网络的流量。

7.12.3　日志迁移

理想的解决方案是不需要将命中传递给服务器的。毕竟，缓存就可以记录下所有的命中。缓存只要将命中日志发送给服务器就行了。实际上，为了保持内容提供商们的满意度，有些大型缓存的提供商已经在对缓存日志进行人工处理，并将其传送给受影响的内容提供商了。 195

但是，命中日志很大，很难移动。而缓存日志并没有被标准化或被组织成独立的日志，以传送给单独的内容提供商。而且，这里面还存在着认证和隐私问题。

已经有一些高效（和不那么高效的）日志分发策略的建议了。但还没有哪个建议成熟到足以为 Web 软件厂商采用。很多建议都非常复杂，需要联合商业伙伴才能实现。[24] 有几家联合厂商已经开始开发广告收入改造工程的支撑框架了。

7.12.4　命中计数和使用限制

RFC 2227，“HTTP 的简单命中计数和使用限制”中定义了一种简单得多的方案。这个协议向 HTTP 中添加了一个称为 `Meter` 的首部，这个首部会周期性地将对特定 URL 的命中次数回送给服务器。通过这种方式，服务器可以从缓存周期性地获取对已缓存文档命中次数的更新。

而且，服务器还能控制在缓存必须向服务器汇报之前，其中的文档还可以使用多少次，或者为缓存文档设置一个时钟超时值。这种控制方式被称为使用限制；通过这种方式，服务器可以对缓存向原始服务器汇报之前，已缓存资源的使用次数进行控制。

我们将在第 21 章详细介绍 RFC 2227。

7.13　更多信息

更多有关缓存的信息，请参见以下参考资源。

- http://www.w3.org/Protocols/rfc2616/rfc2616.txt
  RFC 2616，由 R. Fielding、J. Gettys、J. Mogul、H. Frystyk、L. Mastinter、P. Leach 和 T. Berners-Lee 编写的“Hypertext Transfer Protocol”。

- *Web Caching*（《Web 缓存》）
  Duane Wessels 编写，O’Reilly & Associates 公司出版。

注 24：已经启动了几个商业项目，在尝试开发综合了缓存和日志功能的全球性解决方案。

- http://www.ietf.org/rfc/rfc3040.txt
  RFC 3040，“Internet Web Replication and Caching Taxonomy”（“因特网 Web 复制与缓存分类法”）。

- *Web Proxy Servers*（《Web 代理服务器》）
  Ari Luotonen 编写，Prentice Hall 计算机图书。

- http://www.ietf.org/rfc/rfc3143.txt
  RFC 3143，“Known HTTP Proxy/Caching Problems”（“已知的 HTTP 代理 / 缓存问题”）。

- http://www.squid-cache.org
  196 Squid Web 代理缓存。

第8章

集成点：网关、隧道及中继

事实证明，Web 是一种强大的内容发布工具。随着时间的流逝，人们已经从只在网上发送静态的在线文档，发展到共享更复杂的资源，比如数据库内容或动态生成的 HTML 页面。Web 浏览器这样的 HTTP 应用程序为用户提供了一种统一的方式来访问因特网上的内容。

HTTP 也已成为应用程序开发者的一种基本构造模块，开发者们可以在 HTTP 上捎回其他的协议内容（比如，可以将其他协议的流量包裹在 HTTP 中，用 HTTP 通过隧道或中继方式将这些流量传过公司的防火墙）。Web 上所有的资源都可以使用 HTTP 协议，而且其他应用程序和应用程序协议也可以利用 HTTP 来完成它们的任务。

本章简要介绍了一些开发者用 HTTP 访问不同资源的方法，展示了开发者如何将 HTTP 作为框架启动其他协议和应用程序通信。

本章会讨论：

- 在 HTTP 和其他协议及应用程序之间起到接口作用的网关；
- 允许不同类型的 Web 应用程序互相通信的应用程序接口；
- 允许用户在 HTTP 连接上发送非 HTTP 流量的隧道；
- 作为一种简化的 HTTP 代理，一次将数据转发一跳的中继。

8.1　网关

HTTP 扩展和接口的发展是由用户需求驱动的。要在 Web 上发布更复杂资源的需求
197 出现时，人们很快就明确了一点：单个应用程序无法处理所有这些能想到的资源。

为了解决这个问题，开发者提出了*网关*（gateway）的概念，网关可以作为某种翻译器使用，它抽象出了一种能够到达资源的方法。网关是资源和应用程序之间的粘合剂。应用程序可以（通过 HTTP 或其他已定义的接口）请求网关来处理某条请求，网关可以提供一条响应。网关可以向数据库发送查询语句，或者生成动态的内容，就像一个门一样：进去一条请求，出来一个响应。

图 8-1 显示的是一种资源网关。在这里，Joe 的五金商店服务器就是作为连接数据库内容的网关使用的——注意，客户端只是在通过 HTTP 请求资源，而 Joe 的五金商店的服务器在与网关进行交互以获取资源。

有些网关会自动将 HTTP 流量转换为其他协议，这样 HTTP 客户端无需了解其他协议，就可以与其他应用程序进行交互了（参见图 8-2）。

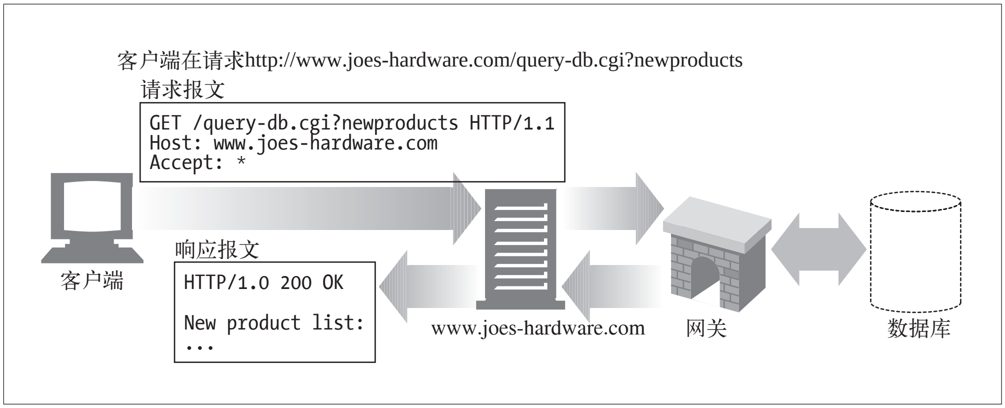


图 8-1　网关的魔力

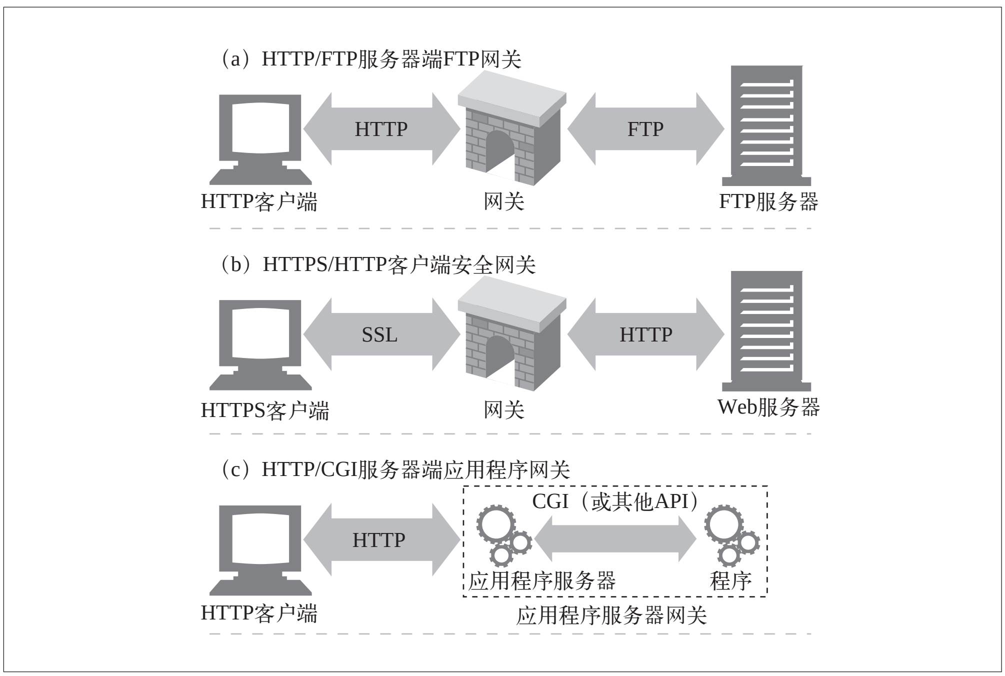


图 8-2　三个 Web 网关实例

图 8-2 显示了三个网关的示例。

- 在图 8-2a 中，网关收到了对 FTP URL 的 HTTP 请求。然后网关打开 FTP 连接，并向 FTP 服务器发布适当的命令。然后将文档和正确的 HTTP 首部通过 HTTP 回送。
- 在图 8-2b 中，网关通过 SSL 收到了一条加密的 Web 请求，网关会对请求进行解

密，[1] 然后向目标服务器转发一条普通的 HTTP 请求。可以将这些安全加速器直
198 接放在（通常处于同一场所的）Web 服务器前面，以便为原始服务器提供高性能的加密机制。

- 在图 8-2c 中，网关通过应用程序服务器网关 API，将 HTTP 客户端连接到服务器端的应用程序上去。在网上的电子商店购物、查看天气预报，或者获取股票报价时，访问的就是应用程序服务器网关。

客户端和服务器端网关

Web 网关在一侧使用 HTTP 协议，在另一侧使用另一种协议。[2]

可以用一个斜杠来分隔客户端和服务器端协议，并以此对网关进行描述：

< 客户端协议 >/< 服务器端协议 >

因此，将 HTTP 客户端连接到 NNTP 新闻服务器的网关就是一个 HTTP/NNTP 网关。我们用术语*服务器端网关*和*客户端网关*来说明对话是在网关的哪一侧进行的。

- *服务器端网关*（server-side gateway）通过 HTTP 与客户端对话，通过其他协议与服务器通信（HTTP/*）。
- *客户端网关*（client-side gateway）通过其他协议与客户端对话，通过 HTTP 与服
199 务器通信（*/HTTP）。

8.2　协议网关

将 HTTP 流量导向网关时所使用的方式与将流量导向代理的方式相同。最常见的方式是，显式地配置浏览器使用网关，对流量进行透明的拦截，或者将网关配置为替代者（反向代理）。

图 8-3 显示了配置浏览器使用服务器端 FTP 网关的对话框。在图中显示的配置中，配置浏览器将 gw1.joes-hardware.com 作为所有 FTP URL 的 HTTP/FTP 网关。浏览器没有将 FTP 命令发送给 FTP 服务器，而是将 HTTP 命令发送给端口 8080 上的 HTTP/FTP 网关 gw1.joes-hardware.com。

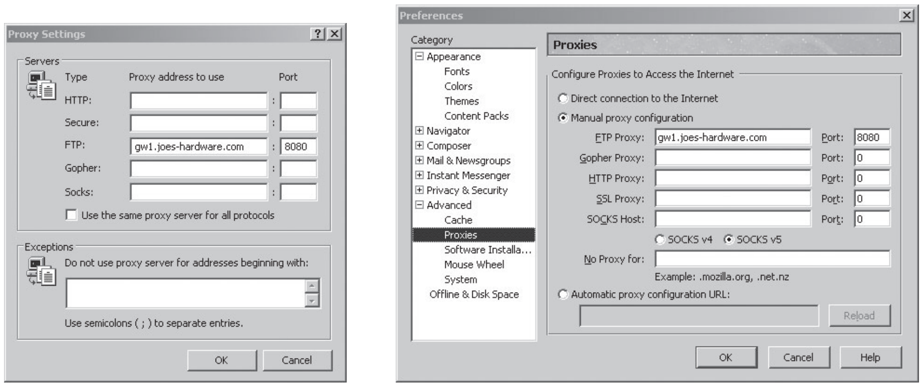
图 8-4 给出了这种网关配置的结果。一般的 HTTP 流量不受影响，会继续流入原始服务器。但对 FTP URL 的请求则被放在 HTTP 请求中发送给网关 gw1.joes-

---

注 1：网关上要安装适当的服务器证书。

注 2：在不同 HTTP 版本之间进行转换的 Web 代理就像网关一样，它们会执行复杂的逻辑，以便在各个端点之间进行沟通。但因为它们在两侧使用的都是 HTTP，所以从技术上来讲，它们还是代理。

hardware.com。网关代表客户端执行 FTP 事务，并通过 HTTP 将结果回送给客户端。



（a）MSIE的手工代理设置　　（b）Navigator的手工代理设置

图 8-3　配置一个 HTTP/FTP 网关

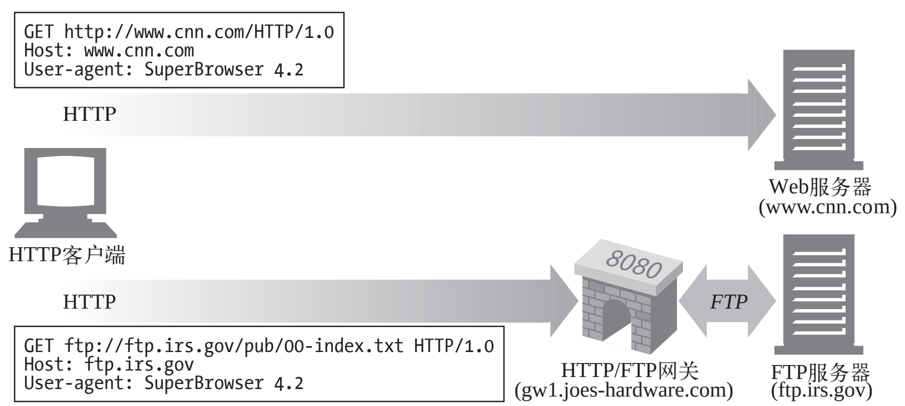


图 8-4　浏览器可以通过配置，让特定的协议使用特定的网关

后面的小节会介绍各种常见网关类型：服务器协议转换器、服务器端安全网关、客户端安全网关以及应用程序服务器。

8.2.1　HTTP/*：服务器端Web网关

请求流入原始服务器时，服务器端 Web 网关会将客户端 HTTP 请求转换为其他协议（参见图 8-5）。

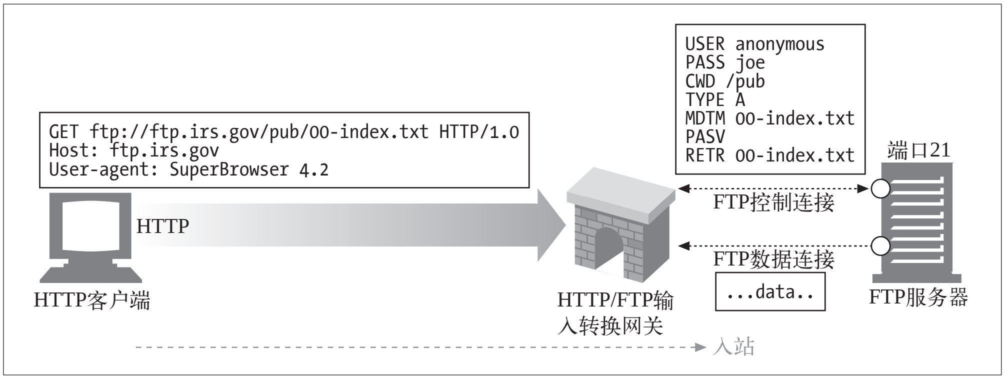


图 8-5　HTTP/FTP 网关将 HTTP 请求转换成 FTP 请求

200 在图 8-5 中，网关收到了一条对 FTP 资源的 HTTP 请求：

ftp://ftp.irs.gov/pub/00-index.txt

网关会打开一条到原始服务器 FTP 端口（端口 21）的 FTP 连接，通过 FTP 协议获取对象。网关会做下列事情：

- 发送 USER 和 PASS 命令登录到服务器上去；
- 发布 CWD 命令，转移到服务器上合适的目录中去；
- 将下载类型设置为 ASCII；
- 用 MDTM 获取文档的最后修改时间；
- 用 PASV 告诉服务器将有被动数据获取请求到达；
- 用 RETR 请求进行对象获取；
- 打开到 FTP 服务器的数据连接，服务器端口由控制信道返回；一旦数据信道打
201 开了，就将对象内容回送给网关。

完成获取之后，会将对象放在一条 HTTP 响应中回送给客户端。

8.2.2　HTTP/HTTPS：服务器端安全网关

一个组织可以通过网关对所有的输入 Web 请求加密，以提供额外的隐私和安全性保护。客户端可以用普通的 HTTP 浏览 Web 内容，但网关会自动加密用户的对话（参见图 8-6）。

8.2.3　HTTPS/HTTP客户端安全加速器网关

最近，将 HTTPS/HTTP 网关作为安全加速器使用的情况是越来越多了。这些 HTTPS/HTTP 网关位于 Web 服务器之前，通常作为不可见的拦截网关或反向代理

使用。它们接收安全的 HTTPS 流量，对安全流量进行解密，并向 Web 服务器发送普通的 HTTP 请求（参见图 8-7）。

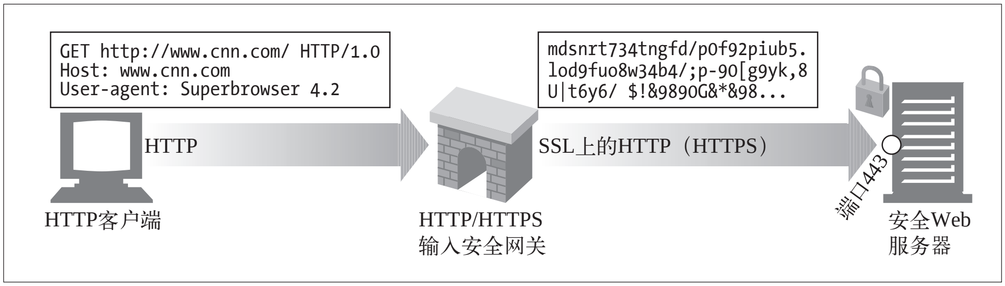


图 8-6　输入 HTTP/HTTPS 安全网关

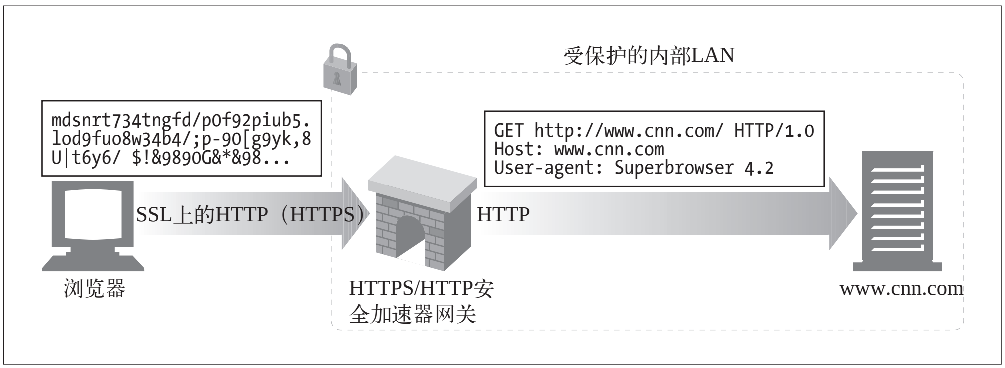


图 8-7　HTTPS/HTTP 安全加速器网关

这些网关中通常都包含专用的解密硬件，以比原始服务器有效得多的方式来解密安全流量，以减轻原始服务器的负荷。这些网关在网关和原始服务器之间发送的是未加密的流量，所以，要谨慎使用，确保网关和原始服务器之间的网络是安全的。 202

8.3　资源网关

到目前为止，我们一直在讨论通过网络连接客户端和服务器的网关。但最常见的网关，应用程序服务器，会将目标服务器与网关结合在一个服务器中实现。应用程序服务器是服务器端网关，与客户端通过 HTTP 进行通信，并与服务器端的应用程序相连（参见图 8-8）。

在图 8-8 中，两个客户端是通过 HTTP 连接到应用程序服务器的。但应用程序服务器并没有回送文件，而是将请求通过一个网关应用编程接口（Application Programming Interface，API）发送给运行在服务器上的应用程序。

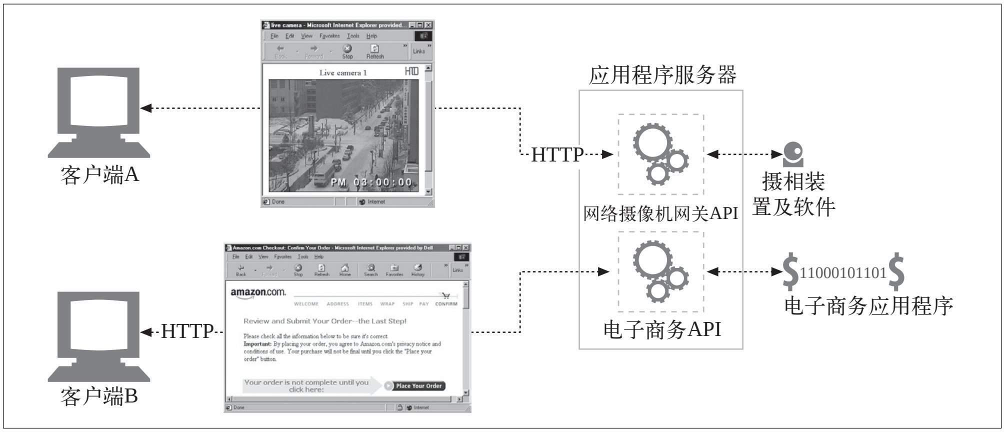


图 8-8　应用程序服务器可以将 HTTP 客户端连接任意后台应用程序

- 收到客户端 A 的请求，根据 URI 将其通过 API 发送给一个数码摄相机应用程序。将得到的图片绑定到一条 HTTP 响应报文中，再回送给客户端，在客户端的浏览器中显示。
- 客户端 B 的 URI 请求的是一个电子商务应用程序。客户端 B 的请求是通过服务器网关 API 发送给电子商务软件的，结果会被回送给浏览器。电子商务软件与客户端进行交互，引导用户通过一系列 HTML 页面来完成购物。

第一个流行的应用程序网关 API 就是*通用网关接口*（Common Gateway Interface，CGI）。CGI 是一个标准接口集，Web 服务器可以用它来装载程序以响应对特定 URL 的 HTTP 请求，并收集程序的输出数据，将其放在 HTTP 响应中回送。在过去的几年中，商业 Web 服务器提供了一些更复杂的接口，以便将 Web 服务器连接到
203 应用程序上去。

早期的 Web 服务器是相当简单的，在网关接口的实现过程中采用的简单方式一直持续到了今天。

请求需要使用网关的资源时，服务器会请辅助应用程序来处理请求。服务器会将辅助应用程序所需的数据传送给它。通常就是整条请求，或者用户想在数据库上运行的请求（来自 URL 的请求字符串，参见第 2 章）之类的东西。

然后，它会向服务器返回一条响应或响应数据，服务器则会将其转发给客户端。服务器和网关是相互独立的应用程序，因此，它们的责任是分得很清楚的。图 8-9 显示了服务器与网关应用程序之间交互的基本运行机制。

这个简单的协议（输入请求，转交，响应）就是最古老，也最常用的服务器扩展接口 CGI 的本质。

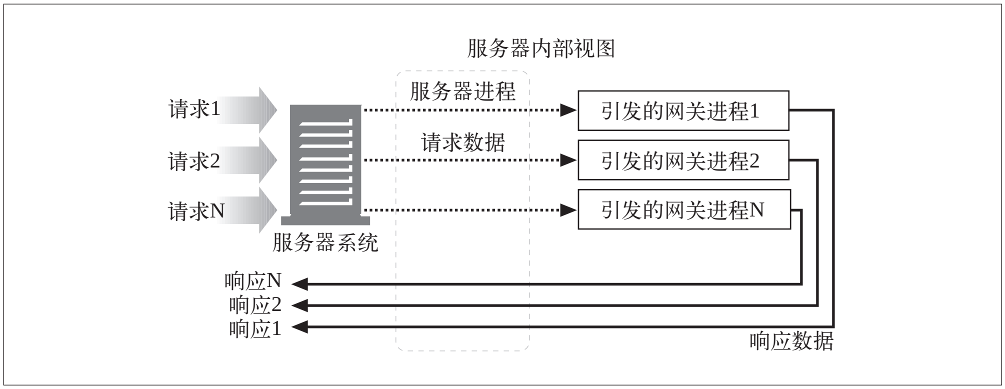


图 8-9　服务器网关应用程序机制

8.3.1　CGI

CGI 是第一个，可能仍然是得到最广泛使用的服务器扩展。在 Web 上广泛用于动态 HTML、信用卡处理以及数据库查询等任务。

CGI 应用程序是独立于服务器的，所以，几乎可以用任意语言来实现，包括 Perl、Tcl、C 和各种 shell 语言。CGI 很简单，几乎所有的 HTTP 服务器都支持它。图 8-9 显示了 CGI 模型的基本运行机制。

CGI 的处理对用户来说是不可见的。从客户端的角度来看，就像发起一个普通请求一样。它完全不清楚服务器和 CGI 应用程序之间的转接过程。URL 中出现字符 cgi 和可能出现的“?”是客户端发现使用了 CGI 应用程序的唯一线索。 204

看来 CGI 是很棒的，对吧？嗯，好吧，既是也不是。它在服务器和众多的资源类型之间提供了一种简单的、函数形式的粘合方式，用来处理各种需要的转换。这个接口还能很好地保护服务器，防止一些糟糕的扩展对它造成的破坏（如果这些扩展直接与服务器相连，造成的错误可能会引发服务器崩溃）。

但是，这种分离会造成性能的耗费。为每条 CGI 请求引发一个新进程的开销是很高的，会限制那些使用 CGI 的服务器的性能，并且会加重服务端机器资源的负担。为了解决这个问题，人们开发了一种新型 CGI——并将其恰当地称为*快速* CGI。这个接口模拟了 CGI，但它是作为持久守护进程运行的，消除了为每个请求建立或拆除新进程所带来的性能损耗。

8.3.2　服务器扩展API

CGI 协议为外部翻译器与现有的 HTTP 服务器提供了一种简洁的接口方式，但如

果想要改变服务器自身的行为，或者只是想尽可能地提升能从服务器上获得的性能呢？服务器开发者为这两种需求提供了几种服务器扩展 API，为 Web 开发者提供了强大的接口，以便他们将自己的模块与 HTTP 服务器直接相连。扩展 API 允许程序员将自己的代码嫁接到服务器上，或者用自己的代码将服务器的一个组件完整地替换出来。

大多数流行的服务器都会为开发者提供一个或多个扩展 API。这些扩展通常都会绑定在服务器自身的结构上，所以，大多数都是某种服务器类型特有的。微软、网景、Apache 和其他服务器都提供了一些 API 接口，允许开发者通过这些接口改变服务器的行为，或者为不同的资源提供一些定制的接口。这些定制接口为开发者提供了强大的接口方式。

微软的 FPSE（FrontPage 服务器端扩展）就是服务器扩展的一个实例，它为使用 FrontPage 的作者进行 Web 发布提供支持。FPSE 能够对 FrontPage 客户端发送的 RPC（remote procedure call，远程过程调用）命令进行解释。这些命令会在 HTTP 中（具体来说，就是在 HTTP POST 方法上）捎回。细节请参见 19.1 节。

8.4 应用程序接口和Web服务

我们已经讨论过可以将资源网关作为 Web 服务器与应用程序的通信方式使用。更广泛地说，随着 Web 应用程序提供的服务类型越来越多，有一点变得越来越清晰了：HTTP 可以作为一种连接应用程序的基础软件来使用。在将应用程序连接起来的过程中，一个更为棘手的问题是在两个应用程序之间进行协议接口的协商，以便这些
205 应用程序可以进行数据的交换——这通常都是针对具体应用程序的个案进行的。

应用程序之间要配合工作，所要交互的信息比 HTTP 首部所能表达的信息要复杂得多。第 19 章描述了几个用于交换定制信息的扩展 HTTP 或 HTTP 上层协议实例。19.1 节介绍的是在 HTTP POST 报文之上建立 RPC 层，19.2 节介绍的是向 HTTP 首部添加 XML 的问题。

因特网委员会开发了一组允许 Web 应用程序之间相互通信的标准和协议。尽管 Web 服务（Web service）可以用来表示独立的 Web 应用程序（构造模块），这里我们还是宽松地用这个术语来表示这些标准。Web 服务的引入并不新鲜，但这是应用程序共享信息的一种新机制。Web 服务是构建在标准的 Web 技术（比如 HTTP）之上的。

Web 服务可以用 XML 通过 SOAP 来交换信息。XML（Extensible Markup Language，扩展标记语言）提供了一种创建数据对象的定制信息，并对其进行解释的方法。SOAP（Simple Object Access Protocol，简单对象访问协议）是向 HTTP 报文中添加

XML 信息的标准方式。[3]

8.5 隧道

我们已经讨论了几种不同的方式，通过这些方式可以用 HTTP 对不同类型的资源进行访问（通过网关），或者是用 HTTP 来启动应用程序到应用程序的通信。在本节中，我们要看看 HTTP 的另一种用法——Web 隧道（Web tunnel），这种方式可以通过 HTTP 应用程序访问使用非 HTTP 协议的应用程序。

Web 隧道允许用户通过 HTTP 连接发送非 HTTP 流量，这样就可以在 HTTP 上捎带其他协议数据了。使用 Web 隧道最常见的原因就是要在 HTTP 连接中嵌入非 HTTP 流量，这样，这类流量就可以穿过只允许 Web 流量通过的防火墙了。

8.5.1 用CONNECT建立HTTP隧道

Web 隧道是用 HTTP 的 CONNECT 方法建立起来的。CONNECT 方法并不是 HTTP/1.1 核心规范的一部分，[4] 但却是一种得到广泛应用的扩展。可以在 Ari Luotonen 的过期因特网草案规范“Tunneling TCP based protocols through Web proxy servers”（“通过 Web 代理服务器用隧道方式传输基于 TCP 的协议”），或他的著作 *Web Proxy Servers* 中找到这些技术规范，本章末尾引用了这两份资源。 206

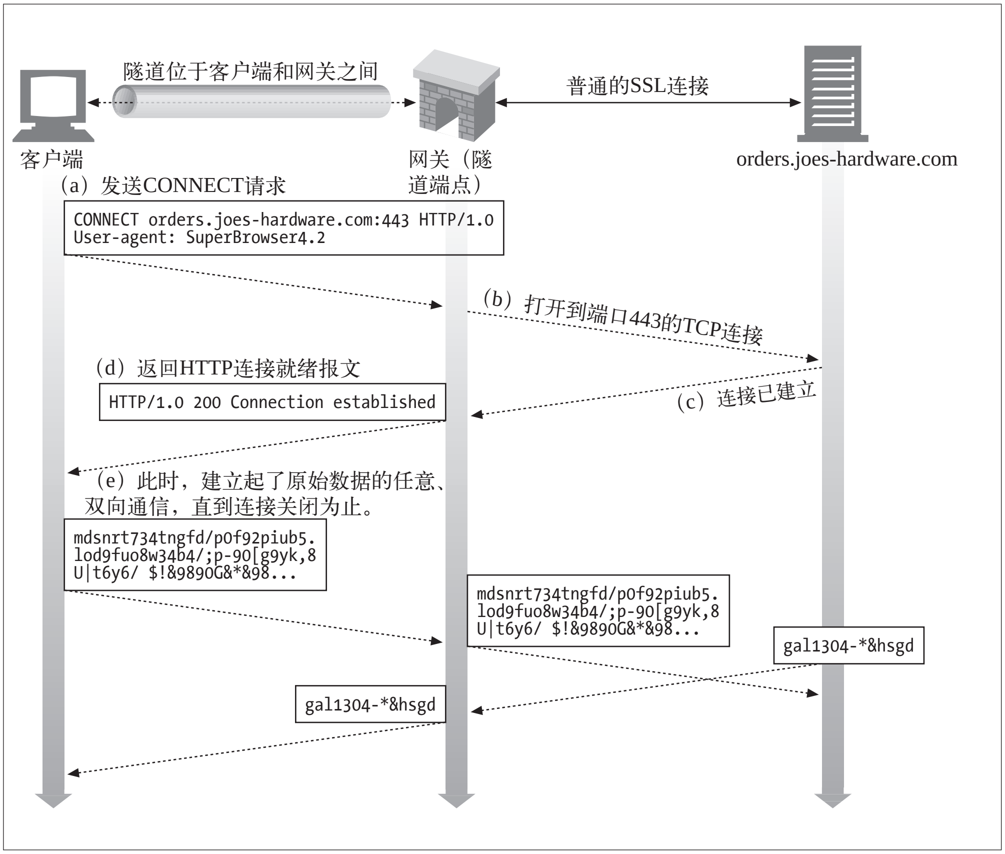
CONNECT 方法请求隧道网关创建一条到达任意目的服务器和端口的 TCP 连接，并对客户端和服务器之间的后继数据进行盲转发。

图 8-10 显示了 CONNECT 方法如何建立起一条到达网关的隧道。

- 在图 8-10a 中，客户端发送了一条 CONNECT 请求给隧道网关。客户端的 CONNECT 方法请求隧道网关打开一条 TCP 连接（在这里，打开的是到主机 orders.joes-hardware.com 的标准 SSL 端口 443 的连接）。
- 在图 8-10b 和图 8-10c 中创建了 TCP 连接。
- 一旦建立了 TCP 连接，网关就会发送一条 HTTP 200 Connection Established 响应来通知客户端（参见图 8-10d）。
- 此时，隧道就建立起来了。客户端通过 HTTP 隧道发送的所有数据都会被直接转发给输出 TCP 连接，服务器发送的所有数据都会通过 HTTP 隧道转发给客户端。

注 3：更多信息，请参见 http://www.w3.org/TR/2001/WD-soap12-part0-20011217/。Doug Tidwell、James Snell 和 Pavel Kulchenko 编写的 *Programming Web Services with SOAP*（SOAP Web 服务开发）一书（O’Reilly）也是非常好的 SOAP 协议信息资源。

注 4：HTTP/1.1 规范保留了 CONNECT 方法，但没有对其功能进行描述。



207 图 8-10　用 CONNECT 建立一条 SSL 隧道

图 8-10 中的例子描述了一条 SSL 隧道，其中的 SSL 流量是在一条 HTTP 连接上发送的，但是通过 CONNECT 方法可以与使用任意协议的任意服务器建立 TCP 连接的。

1. CONNECT请求

除了起始行之外，CONNECT 的语法与其他 HTTP 方法类似。一个后面跟着冒号和端口号的主机名取代了请求 URI。主机和端口都必须指定：

```
CONNECT home.netscape.com:443 HTTP/1.0
User-agent: Mozilla/4.0
```

和其他 HTTP 报文一样，起始行之后，有零个或多个 HTTP 请求首部字段。这些行照例以 CRLF 结尾，首部列表以一个仅有 CRLF 的空行结束。

2. CONNECT响应

发送了请求之后，客户端会等待来自网关的响应。和普通 HTTP 报文一样，响应码 200 表示成功。按照惯例，响应中的原因短语通常被设置为“Connection Established”：

```
HTTP/1.0 200 Connection Established
Proxy-agent: Netscape-Proxy/1.1
```

与普通 HTTP 响应不同，这个响应并不需要包含 `Content-Type` 首部。此时连接只是对原始字节进行转接，不再是报文的承载者，所以不需要使用内容类型了。[5]

8.5.2 数据隧道、定时及连接管理

管道化数据对网关是不透明的，所以网关不能对分组的顺序和分组流作任何假设。一旦隧道建立起来了，数据就可以在任意时间流向任意方向了。[6]

作为一种性能优化方法，允许客户端在发送了 CONNECT 请求之后，接收响应之前，发送隧道数据。这样可以更快地将数据发送给服务器，但这就意味着网关必须能够正确处理跟在请求之后的数据。尤其是，网关不能假设网络 I/O 请求只会返回首部数据，网关必须确保在连接准备就绪时，将与首部一同读进来的数据发送给服 208
务器。在请求之后以管道方式发送数据的客户端，如果发现回送的响应是认证请求，或者其他非 200 但不致命的错误状态，就必须做好重发请求数据的准备。[7]

如果在任意时刻，隧道的任意一个端点断开了连接，那个端点发出的所有未传输数据都会被传送给另一个端点，之后，到另一个端点的连接也会被代理终止。如果还有数据要传输给关闭连接的端点，数据会被丢弃。

8.5.3 SSL隧道

最初开发 Web 隧道是为了通过防火墙来传输加密的 SSL 流量。很多组织都会将所有流量通过分组过滤路由器和代理服务器以隧道方式传输，以提升安全性。但有些协议，比如加密 SSL，其信息是加密的，无法通过传统的代理服务器转发。隧道会通过一条 HTTP 连接来传输 SSL 流量，以穿过端口 80 的 HTTP 防火墙（参见图 8-11）。

注 5：为了实现一致性，今后的规范可能会为隧道定义一个媒体类型（比如 application/tunnel）。

注 6：隧道的两端（客户端和网关）必须做好在任意时刻接收来自连接任一端分组的准备，而且必须将数据立即转发出去。由于隧道化协议中可能包含了数据的依赖关系，所以隧道的任一端都不能忽略输入数据。隧道一端对数据的消耗不足可能会将隧道另一端的数据生产者挂起，造成死锁。

注 7：传送的数据不要超过请求 TCP 分组的剩余容量。如果在收到所有管道化传输的 TCP 分组之前，网关关闭了连接，那么，管道化传输的多余数据就会使客户端 TCP 重置。TCP 重置会使客户端丢失收到的网关响应，这样客户端就无法分辨错误是由于网络错误、访问控制，还是认证请求造成的了。

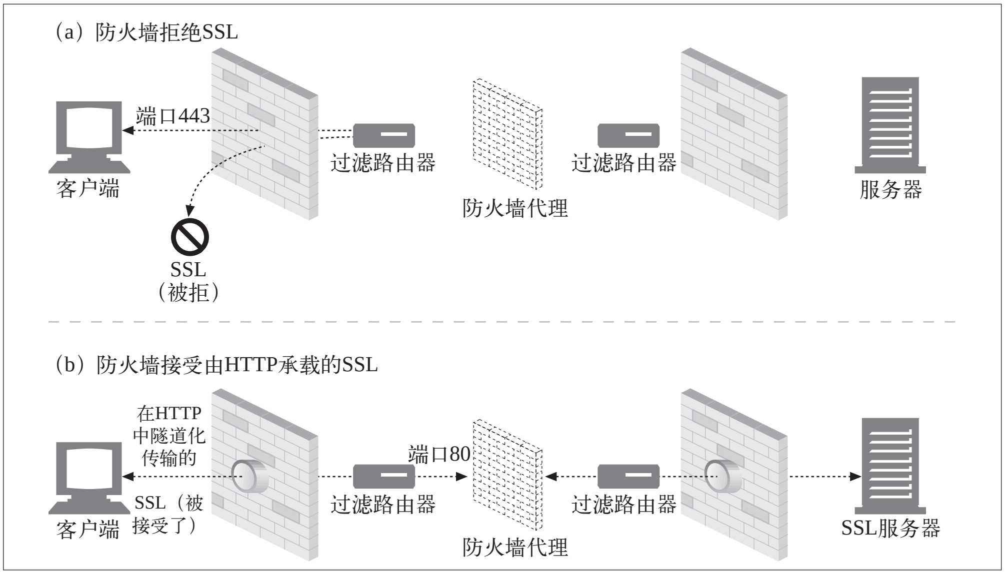


图 8-11　隧道可以经由 HTTP 连接传输非 HTTP 流量

为了让 SSL 流量经现存的代理防火墙进行传输，HTTP 中添加了一项隧道特性，在此特性中，可以将原始的加密数据放在 HTTP 报文中，通过普通的 HTTP 信道传送

209 （参见图 8-12）。

在图 8-12a 中，SSL 流量被直接发送给了一个（SSL 端口 443 上的）安全 Web 服务器。在图 8-12b 中，SSL 流量被封装到一条 HTTP 报文中，并通过 HTTP 端口 80 上的连接发送，最后被解封装为普通的 SSL 连接。

通常会用隧道将非 HTTP 流量传过端口过滤防火墙。这一点可以得到很好的利用，比如，通过防火墙传输安全 SSL 流量。但是，这项特性可能会被滥用，使得恶意协议通过 HTTP 隧道流入某个组织内部。

8.5.4　SSL隧道与HTTP/HTTPS网关的对比

可以像其他协议一样，对 HTTPS 协议（SSL 上的 HTTP）进行网关操作：由网关（而不是客户端）初始化与远端 HTTPS 服务器的 SSL 会话，然后代表客户端执行 HTTPS 事务。响应会由代理接收并解密，然后通过（不安全的）HTTP 传送给客户端。这是网关处理 FTP 的方式。但这种方式有几个缺点：

- 客户端到网关之间的连接是普通的非安全 HTTP；
- 尽管代理是已认证主体，但客户端无法对远端服务器执行 SSL 客户端认证（基于 X509 证书的认证）；

- 网关要支持完整的 SSL 实现。 210

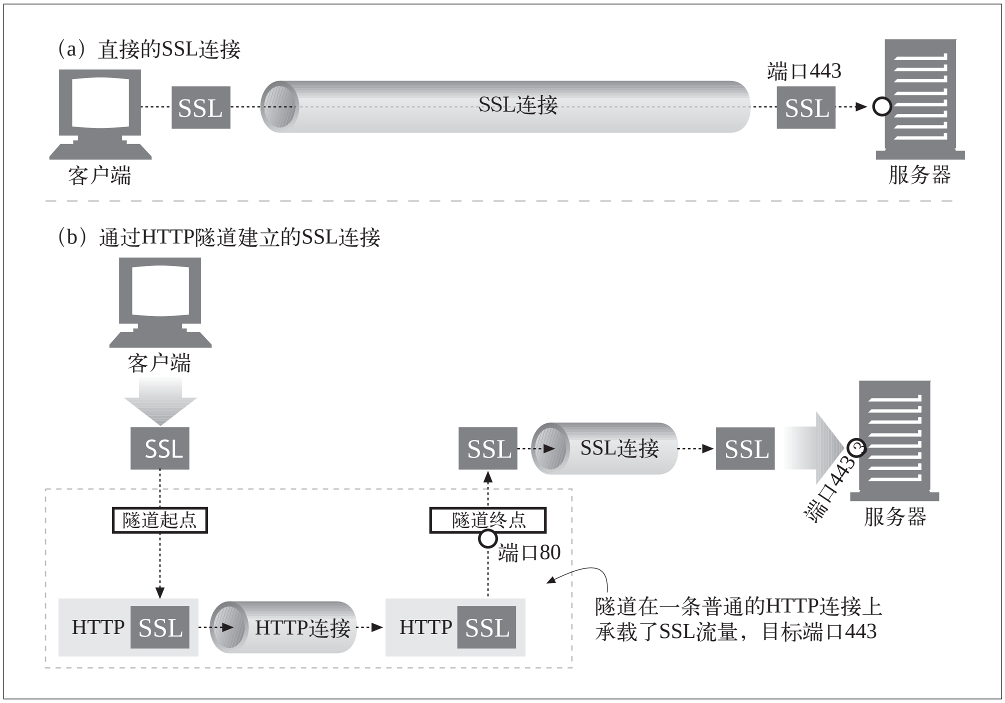


图 8-12　直接的 SSL 连接与隧道化 SSL 连接的对比

注意，对于 SSL 隧道机制来说，无需在代理中实现 SSL。SSL 会话是建立在产生请求的客户端和目的（安全的）Web 服务器之间的，中间的代理服务器只是将加密数据经过隧道传输，并不会在安全事务中扮演其他的角色。

8.5.5　隧道认证

在适当的情况下，也可以将 HTTP 的其他特性与隧道配合使用。尤其是，可以将代理的认证支持与隧道配合使用，对客户端使用隧道的权利进行认证（参见图 8-13）。

8.5.6　隧道的安全性考虑

总的来说，隧道网关无法验证目前使用的协议是否就是它原本打算经过隧道传输的协议。因此，比如说，一些喜欢捣乱的用户可能会通过本打算用于 SSL 的隧道，越 211
过公司防火墙传递因特网游戏流量，而恶意用户可能会用隧道打开 Telnet 会话，或用隧道绕过公司的 E-mail 扫描器来发送 E-mail。

为了降低对隧道的滥用，网关应该只为特定的知名端口，比如 HTTPS 的端口 443，打开隧道。

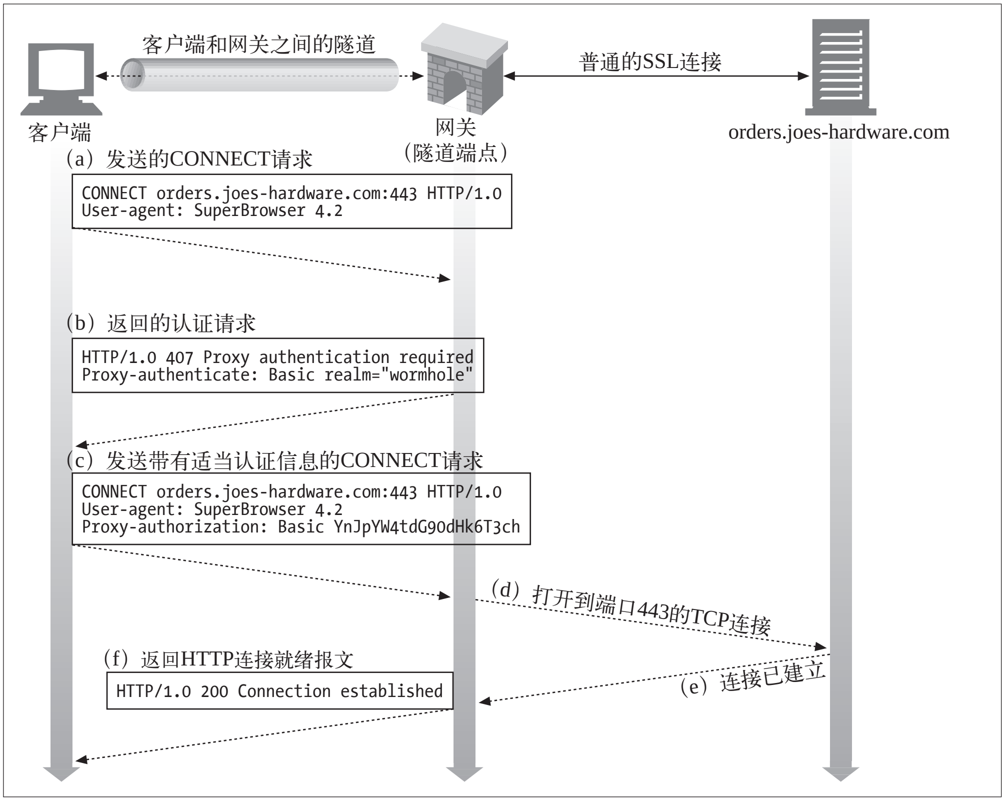


图 8-13　网关允许某客户端使用隧道之前，可以对其进行代理认证

8.6　中继

HTTP 中继（relay）是没有完全遵循 HTTP 规范的简单 HTTP 代理。中继负责处理 HTTP 中建立连接的部分，然后对字节进行盲转发。

HTTP 很复杂，所以实现基本的代理功能并对流量进行盲转发，而且不执行任何首部和方法逻辑，有时是很有用的。盲中继很容易实现，所以有时会提供简单的过滤、诊断或内容转换功能。但这种方式可能潜在严重的互操作问题，所以部署的时候要特别小心。

某些简单盲中继实现中存在的一个更常见（也更声名狼藉的）问题是，由于它们无法正确处理 `Connection` 首部，所以有潜在的挂起 keep-alive 连接的可能。图 8-14 对这种情况进行了说明。

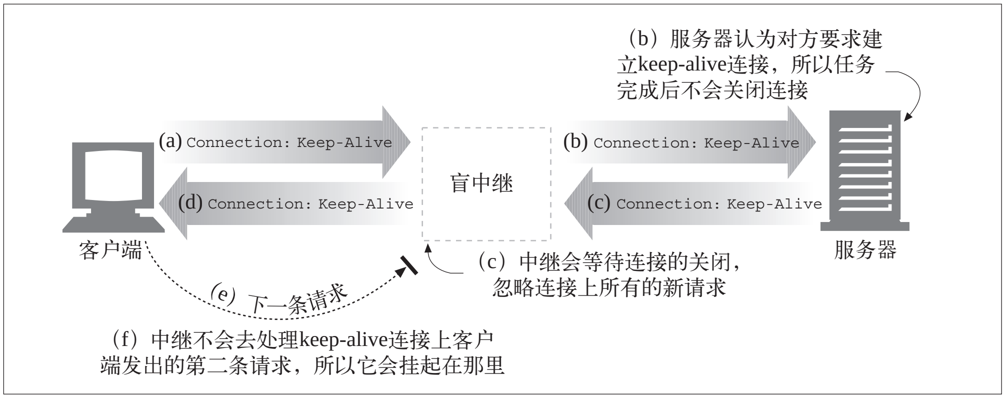


图 8-14　如果简单盲中继是单任务的，且不支持 Connection 首部，就会挂起

这张图中发生的情况如下所述。

- 在图 8-14a 中，Web 客户端向中继发送了一条包含 `Connection: Keep-Alive` 首部的报文，如果可能的话要求建立一条 keep-alive 连接。客户端等待响应，以确定它要求建立 keep-alive 信道的请求是否被认可了。
- 中继收到了这条 HTTP 请求，但它并不理解 `Connection` 首部，因此会将报文一字不漏地沿着链路传递给服务器（参见图 8-14b）。但 `Connection` 首部是个逐跳首部；只适用于单条传输链路，是不应该沿着链路传送下去的。要有不好的事 212 情发生了！
- 在图 8-14b 中，经过中继转发的 HTTP 请求抵达 Web 服务器。当 Web 服务器收到经过代理转发的 `Connection: Keep-Alive` 首部时，会错误地认为中继（对服务器来说，它看起来就和其他客户端一样）要求进行 keep-alive 的对话！这对 Web 服务器来说没什么问题——它同意进行 keep-alive 对话，并在图 8-14c 中回送了一个 `Connection: Keep-Alive` 响应首部。那么，此时，Web 服务器就认为它是在与中继进行 keep-alive 对话，会遵循 keep-alive 对话的规则。但中继对 keep-alive 会话根本就一无所知。
- 在图 8-14d 中，中继将 Web 服务器的响应报文，以及来自 Web 服务器的 `Connection: Keep-Alive` 首部一起发回给客户端。客户端看到这个首部，认为中继同意进行 keep-alive 对话。此时，客户端和服务器都认为它们是在进行 keep-alive 对话，但与它们进行对话的中继却根本不知道什么 keep-alive 对话。
- 中继对持久对话一无所知，所以它会将收到的所有数据都转发给客户端，等待原始服务器关闭连接。但原始服务器认为中继要求服务器将连接保持在活跃状态，所以是不会关闭连接的！这样，中继就会挂起，等待连接的关闭。
- 在图 8-14d 中，当客户端收到回送的响应报文时，它会直接转向第二条请求，在 keep-alive 连接上向中继发送另一条请求（参见图 8-14e）。简单中继通常不会期

待同一条连接上还会有另一条请求到达。浏览器上的圈会不停地转，但没有任何进展。

有一些方法可以使中继稍微智能一些，以消除这些风险，但所有简化的代理都存在着出现互操作性问题的风险。要为某个特定目标构建简单的 HTTP 中继，一定要特别注意其使用方法。对任何大规模部署来说，都要非常认真地考虑使用真正的、完全遵循 HTTP 的代理服务器。

更多与中继和连接管理有关的信息，参见 4.5.6 节。

8.7 更多信息

更多信息，请参见下列参考材料。

- http://www.w3.org/Protocols/rfc2616/rfc2616.txt
  RFC 2616， 由 R. Fielding、J. Gettys、J. Mogul、H. Frystyk、L. Mastinter、P. Leach 和 T. Berners-Lee 编写的“Hypertext Transfer Protocol”。

- *Web Proxy Servers*（《Web 代理服务器》）
  213 Ari Luotonen，Prentice Hall 出版的计算机图书。

- http://www.alternic.org/drafts/drafts-l-m/draft-luotonen-Web-proxy-tunneling-01.txt
  Ari Luotonen 编写的“Tunneling TCP based protocols through Web proxy servers”（“用隧道方式通过 Web 代理服务器传输基于 TCP 的协议”）。

- http://cgi-spec.golux.com
  通用网关接口——RFC 项目页面。

- http://www.w3.org/TR/2001/WD-soap12-part0-20011217/
  W3C——SOAP 版本 1.2 工作草案。

- *Programming Web Services with Soap*[8]（《Soap Web 服务开发》）
  James Snell、Doug Tidwell 和 Pavel Kulchenko 编写，O’Reilly & Associates 公司出版。

- http://www.w3.org/TR/2002/WD-wsa-reqs-20020429
  W3C——Web 服务的架构要求。

- *Web Services Essentials*[9]（《Web 服务精髓》）
  214 Ethan Cerami，O’Reilly & Associates 公司出版。

注 8~9：这二本书的中文版已由中国电力出版社出版。（编者注）

第9章

Web机器人

本章我们来仔细了解一下被称为 Web 机器人（Web robot）的自活跃（self-animating）用户代理，以继续我们的 HTTP 架构之旅。

Web 机器人是能够在无需人类干预的情况下自动进行一系列 Web 事务处理的软件程序。很多机器人会从一个 Web 站点逛到另一个 Web 站点，获取内容，跟踪超链，并对它们找到的数据进行处理。根据这些机器人自动探查 Web 站点的方式，人们为它们起了一些各具特色的名字，比如“爬虫”、“蜘蛛”、“蠕虫”以及“机器人”等，就好像它们都有自己的头脑一样。

这里有几个 Web 机器人的示例。

- 股票图形机器人每隔几分钟就会向股票市场的服务器发送 HTTP GET，用得到的数据来构建股市价格趋势图。
- Web 统计机器人会收集与万维网规模及发展有关的“统计”信息。它们会在 Web 上游荡，统计页面的数量，记录每个页面的大小、所用语言以及媒体类型。[1]
- 搜索引擎机器人会搜集它们所找到的所有文档，以创建搜索数据库。
- 比较购物机器人会从在线商店的目录中收集 Web 页面，构建商品及其价格的数据库。

9.1 爬虫及爬行方式

Web 爬虫是一种机器人，它们会递归地对各种信息性 Web 站点进行遍历，获取第一个 Web 页面，然后获取那个页面指向的所有 Web 页面，然后是那些页面指向的所有 Web 页面，依此类推。递归地追踪这些 Web 链接的机器人会沿着 HTML 超链
215 创建的网络“爬行”，所以将其称为*爬虫*（crawler）或*蜘蛛*（spider）。

因特网搜索引擎使用爬虫在 Web 上游荡，并把它们碰到的文档全部拉回来。然后对这些文档进行处理，形成一个可搜索的数据库，以便用户查找包含了特定单词的文档。网上有数万亿的 Web 页面需要查找和取回，这些搜索引擎蜘蛛必然是些最复杂的机器人。我们来进一步仔细地看看这些爬虫是怎样工作的。

9.1.1 从哪儿开始：根集

在把饥饿的爬虫放出去之前，需要给它一个起始点。爬虫开始访问的 URL 初始集合被称作*根集*（root set）。挑选根集时，应该从足够多不同的站点中选择 URL，这样，爬遍所有的链接才能最终到达大部分你感兴趣的 Web 页面。

注 1：http://www.netcraft.com 收集了大量统计度量值，用于统计 Web 站点使用的是哪种类型的服务器。

要在图 9-1 所示的 Web 上爬行，使用哪个根集比较好呢？与在真实的 Web 中一样，没有哪个文档最终可以链接到所有其他文档上去。如果从图 9-1 的文档 A 开始，可以到达 B、C 和 D，然后是 E 和 F，然后到 J，然后到 K。但没有从 A 到 G，或从 A 到 N 的链路。

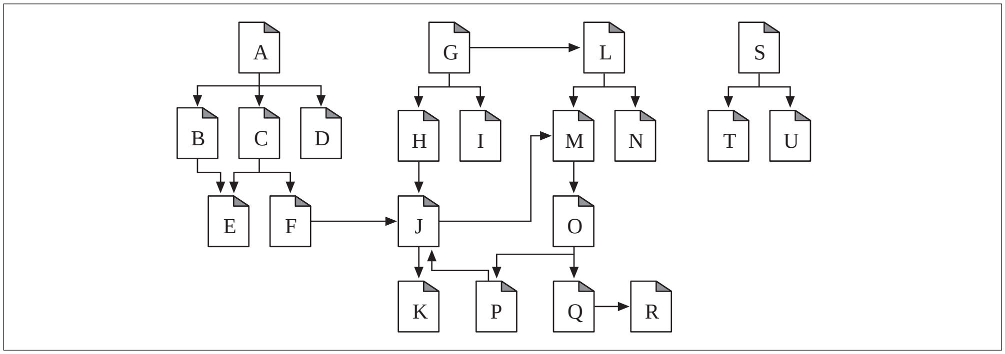


图 9-1　根集要能够到达所有的页面

这个 Web 结构中的某些 Web 页面，比如 S、T 和 U，几乎是被隔离开来的——它们是孤立的，没有任何链接指向它们。可能这些孤独的页面是一些新页面，还没人找到它们。或者可能是一些非常老的或不显眼的页面。

总之，根集中并不需要有很多页面，就可以涵盖一大片 Web 结构。在图 9-1 中，要抵达所有页面，根集中只需要有 A、G 和 S 就行了。

通常，一个好的根集会包括一些大的流行 Web 站点（比如 http://www.yahoo.com）、一个新创建页面的列表和一个不经常被链接的无名页面列表。很多大规模的爬虫产品，比如因特网搜索引擎使用的那些爬虫，都为用户提供了向根集中提交新页面或无名页面的方式。这个根集会随时间推移而增长，是所有新爬虫的种子列表。 216

9.1.2　链接的提取以及相对链接的标准化

爬虫在 Web 上移动时，会不停地对 HTML 页面进行解析。它要对所解析的每个页面上的 URL 链接进行分析，并将这些链接添加到需要爬行的页面列表中去。随着爬虫的前进，当其发现需要探查的新链接时，这个列表常常会迅速地扩张。[2] 爬虫要通过简单的 HTML 解析，将这些链接提取出来，并将相对 URL 转换为绝对形式。2.3.1 节讨论了如何进行这种转换。

注 2：我们会在 9.1.3 节开始讨论爬虫是否需要记住它们到过何处。在爬行过程中，这个已发现 URL 列表会不断扩张，直到已经对 Web 空间进行了彻底的探查为止，这时爬虫就会到达一个不再发现新链接的状态了。

9.1.3　避免环路的出现

机器人在 Web 上爬行时，要特别小心不要陷入循环，或环路（cycle）之中。我们来看看图 9-2 中所示的爬虫。

- 在图 9-2a 中，机器人获取页面 A，看到 A 链接到 B，就获取页面 B。
- 在图 9-2b 中，机器人获取页面 B，看到 B 链接到 C，就获取页面 C。
- 在图 9-2c 中，机器人获取页面 C，会看到 C 链接到 A。如果机器人再次获取页面 A，就会陷入一个环路中，获取 A、B、C、A、B、C、A……

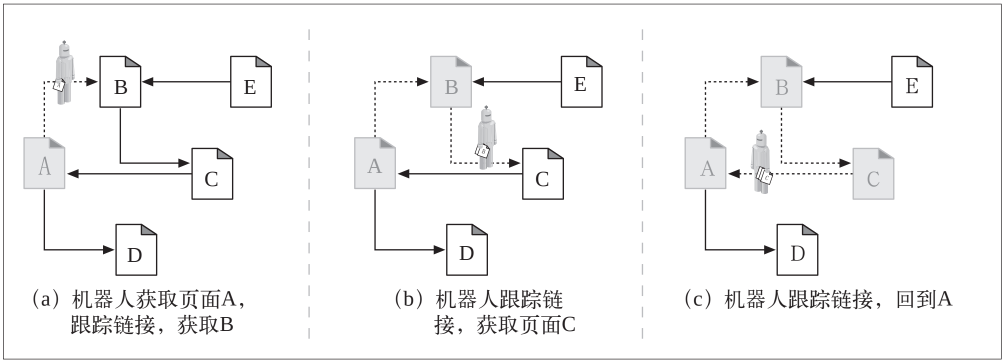


图 9-2　在 Web 的超链中爬行

机器人必须知道它们到过何处，以避免环路的出现。环路会造成机器人陷阱，这些陷阱会暂停或减缓机器人的爬行进程。

9.1.4　循环与复制

至少出于下列三个原因，环路对爬虫来说是有害的。

- 它们会使爬虫陷入可能会将其困住的循环之中。循环会使未经良好设计的爬虫不停地兜圈子，把所有时间都耗费在不停地获取相同的页面上。爬虫会消耗掉很多网络带宽，可能完全无法获取任何其他页面了。
217
- 爬虫不断地获取相同的页面时，另一端的 Web 服务器也在遭受着打击。如果爬虫与服务器连接良好，它就会击垮 Web 站点，阻止所有真实用户访问这个站点。这种拒绝服务是可以作为法律诉讼理由的。
- 即使循环自身不是什么问题，爬虫也是在获取大量重复的页面［通常被称为“dups”（重复），以便与“loops”（循环）押韵］。爬虫应用程序会被重复的内容所充斥，这样应用程序就会变得毫无用处。返回数百份完全相同页面的因特网搜索引擎就是一个这样的例子。

9.1.5 面包屑留下的痕迹

但是，记录曾经到过哪些地方并不总是一件容易的事。编写本书时，因特网上有数十亿个不同的 Web 页面，其中还不包括那些由动态网关产生的内容。

如果要爬行世界范围内的一大块 Web 内容，就要做好访问数十亿 URL 的准备。记录下哪些 URL 已经访问过了是件很具挑战的事情。由于 URL 的数量巨大，所以，要使用复杂的数据结构以便快速判定哪些 URL 是曾经访问过的。数据结构在访问速度和内存使用方面都应该是非常高效的。

数亿 URL 需要具备快速搜索结构，所以速度是很重要的。穷举搜索 URL 列表是根本不可能的。机器人至少要用到搜索树或散列表，以快速判定某个 URL 是否被访问过。

数亿 URL 还会占用大量的空间。如果平均每个 URL 有 40 个字符长，而且一个 Web 机器人要爬行 5 亿个 URL（只是 Web 的一小部分），那么搜索数据结构只是存储这些 URL 就需要 20GB 或更多的存储空间（40 字节 /URL × 5 亿个 URL = 20GB）！

这里列出了大规模 Web 爬虫对其访问过的地址进行管理时使用的一些有用的技术。

- 树和散列表

  复杂的机器人可能会用搜索树或散列表来记录已访问的 URL。这些是加速 URL 查找的软件数据结构。

- 有损的存在位图

  为了减小空间，一些大型爬虫会使用有损数据结构，比如存在位数组（presence bit array）。用一个散列函数将每个 URL 都转换成一个定长的数字，这个数字在数组中有个相关的“存在位”。爬行过一个 URL 时，就将相应的“存在位”置位。如果存在位已经置位了，爬虫就认为已经爬行过那个 URL 了。[3] 218

- 检查点

  一定要将已访问 URL 列表保存到硬盘上，以防机器人程序崩溃。

- 分类

  随着 Web 的扩展，在一台计算机上通过单个机器人来完成爬行就变得不太现实了。那台计算机可能没有足够的内存、磁盘空间、计算能力，或网络带宽来完成爬行任务。

注 3：由于 URL 的潜在数量是无限的，而存在位数组中的比特数是有限的，所以有出现冲突的可能——两个 URL 可能会映射到同一个存在位上去。出现这种情况时，爬虫会错误地认为它已经爬行过某个实际未爬行过的页面了。在实际应用中，使用大量的存在位就可以使这种情况极少发生。产生冲突的后果就是爬虫会忽略某个页面。

有些大型 Web 机器人会使用机器人“集群”，每个独立的计算机是一个机器人，以汇接方式工作。为每个机器人分配一个特定的 URL“片”，由其负责爬行。这些机器人配合工作，爬行整个 Web。机器人个体之间可能需要相互通信，来回传送 URL，以覆盖出故障的对等实体的爬行范围，或协调其工作。

Witten 等人编写的 *Managing Gigabytes: Compressing and Indexing Documents and Images*（《海量数据管理——文档和图像的压缩与索引》）[4]，*Morgan Kaufmann* 出版社出版，是实现大规模数据结构的很好的参考书。这本书讲的全是管理大量数据所需的各种诀窍和技巧。

9.1.6　别名与机器人环路

由于 URL“别名”的存在，即使使用了正确的数据结构，有时也很难分辨出以前是否访问过某个页面。如果两个 URL 看起来不一样，但实际指向的是同一资源，就称这两个 URL 互为“别名”。

表 9-1 列出了不同 URL 指向同一资源的几种简单方式。

表9-1　同一文档的不同URL别名

	第一个URL	第二个URL	什么时候是别名
a	http://www.foo.com/bar.html	http://www.foo.com:80/bar.html	默认端口为 80
b	http://www.foo.com/~fred	http://www.foo.com/%7Ffred	%7F 与 ~ 相同
c	http://www.foo.com/x.html#early	http://www.foo.com/x.html#middle	标签并没有修改页面内容
d	http://www.foo.com/readme.htm	http://www.foo.com/README.HTM	服务器是大小写无关的
e	http://www.foo.com/	http://www.foo.com/index.html	默认页面为 index.html
219 f	http://www.foo.com/index.html	http://209.231.87.45/index.html	www.foo.com 使用这个 IP 地址

9.1.7　规范化URL

大多数 Web 机器人都试图通过将 URL“规范化”为标准格式来消除上面那些显而易见的别名。机器人首先可先通过下列步骤将每个 URL 都转化为规范化的格式。

(1) 如果没有指定端口的话，就向主机名中添加“:80”。
(2) 将所有转义符 %xx 都转换成等价字符。
(3) 删除 # 标签。

注 4：本书中文版已由科学出版社出版。（编者注）

经过这些步骤就可以消除表 9-1 中 a~c 所列的别名问题了。但如果不知道特定 Web 服务器的相关信息，机器人就没什么好办法来避免表 9-1 中 d~f 的问题了。

- 机器人需要知道 Web 服务器是否是大小写无关的才能避免表 9-1d 中的别名问题。
- 机器人需要知道 Web 服务器上这个目录下的索引页面配置才能知道表 9-1e 中的情况是否是别名。
- 即使机器人知道表 9-1f 中的主机名和 IP 地址都指向同一台计算机，它也还要知道 Web 服务器是否配置为进行（参见第 5 章）虚拟主机操作，才能知道这个 URL 是不是别名。

URL 规范化可以消除一些基本的语法别名，但机器人还会遇到其他的、将 URL 转换为标准形式也无法消除的 URL 别名。

9.1.8　文件系统连接环路

文件系统中的符号连接会造成特定的潜在环路，因为它们会在目录层次深度有限的情况下，造成深度无限的假象。符号连接环路通常是由服务器管理员的无心错误造成的，但“邪恶的网管”也可能会恶意地为机器人制造这样的陷阱。

图 9-3 显示了两个文件系统。在图 9-3a 中，subdir 是个普通的目录。在图 9-3b 中，subdir 是个指回到“/”的符号连接。在这两个图中，都假设文件 /index.html 中包含了一个指向文件 subdir/index.html 的超链接。

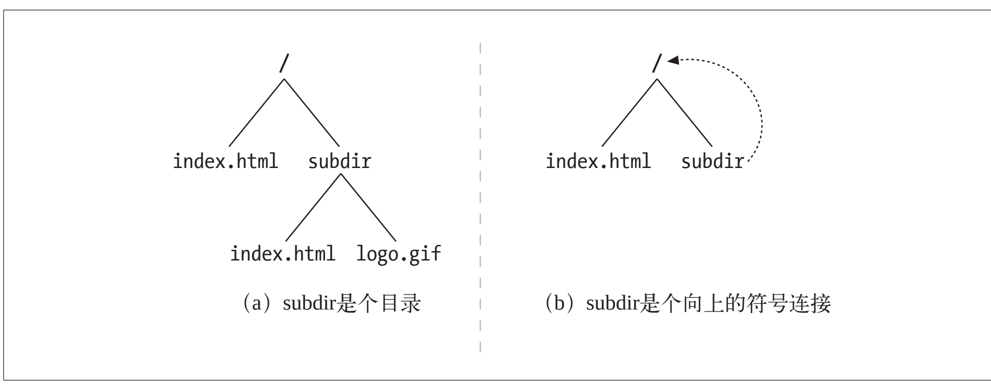


图 9-3　符号连接环路

使用图 9-3a 所示的文件系统时，Web 爬虫可能会采取下列动作：

(1) GET http://www.foo.com/index.html
　获取 /index.html，找到指向 subdir/index.html 的链接。

(2) GET http://www.foo.com/subdir/index.html

获取 subdir/index.html，找到指向 subdir/logo.gif 的链接。

(3) GET http://www.foo.com/subdir/logo.gif

220 获取 subdir/logo.gif，再没有链接了，结束。

但在图 9-3b 的文件系统中，可能会发生下列情况：

(1) GET http://www.foo.com/index.html

获取 /index.html，找到指向 subdir/index.html 的链接。

(2) GET http://www.foo.com/subdir/index.html

获取 subdir/index.html，但得到的还是同一个 index.html。

(3) GET http://www.foo.com/subdir/subdir/index.html

获取 subdir/subdir/index.html。

(4) GET http://www.foo.com/subdir/subdir/subdir/index.html

获取 subdir/subdir/subdir/index.html。

图 9-3b 的问题是 subdir/ 是个指向“/”的环路，但由于 URL 看起来有所不同，所以机器人无法单从 URL 本身判断出文档是相同的。毫无戒备的机器人就有了陷入循环的危险。如果没有某种循环检测方式，这个环路就会继续下去，通常会持续到 URL 的长度超过机器人或服务器的限制为止。

9.1.9　动态虚拟Web空间

恶意网管可能会有意创建一些复杂的爬虫循环来陷害那些无辜的、毫无戒备的机器人。尤其是，发布一个看起来像普通文件，实际上却是网关应用程序的 URL 是很容易的。这个应用程序可以在传输中构造出包含了到同一服务器上虚构 URL 链接的 HTML。请求这些虚构的 URL 时，这个邪恶的服务器就会捏造出一个带有新的虚构 URL 的新 HTML 页面来。

即使这个恶意 Web 服务器实际上并不包含任何文件，它也可以通过无限虚拟的 Web 空间将可怜的机器人带入爱丽丝漫游仙境之旅。更糟的是，每次的 URL 和 HTML 看起来都有很大的不同，机器人很难检测到环路。图 9-4 显示了一个恶意 Web 服务
221 器生成假内容的例子。

更常见的情况是，那些没有恶意的网管们可能会在无意中通过符号连接或动态内容构造出爬虫陷阱。比如，我们来看一个基于 CGI 的日历程序，它会生成一个月历和

一个指向下个月的链接。真正的用户是不会不停地请求下个月的链接的，但不了解其内容的动态特性的机器人可能会不断向这些资源发出无穷的请求。[5]

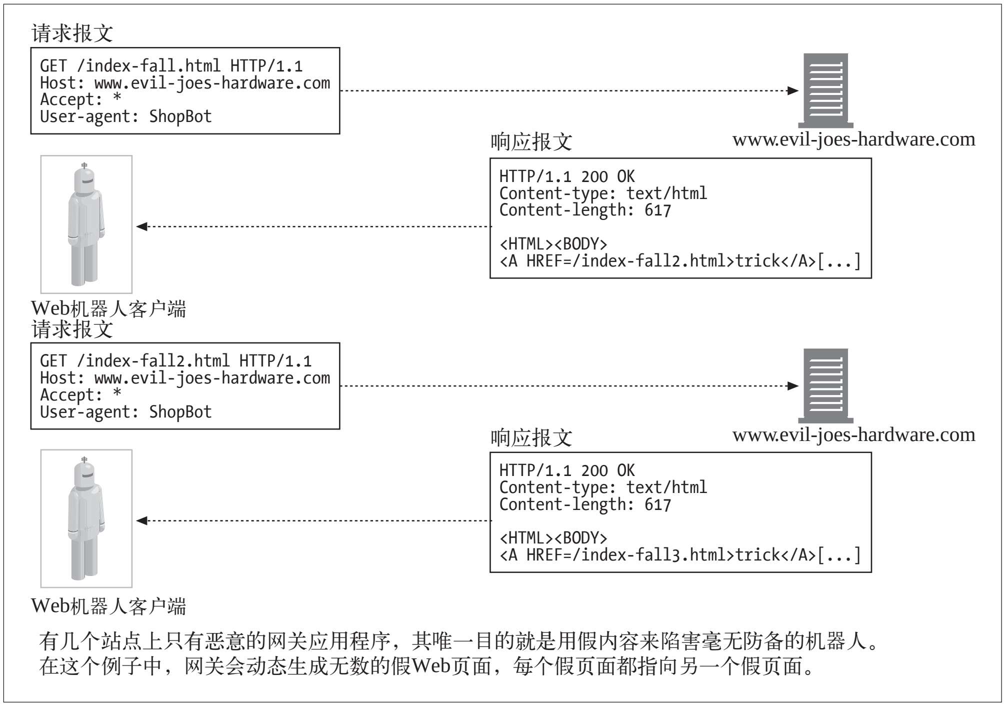


图 9-4　恶意的动态 Web 空间示例

9.1.10　避免循环和重复

没有什么简单明了的方式可以避免所有的环路。实际上，经过良好设计的机器人中要包含一组试探方式，以避免环路的出现。

总的说来，爬虫的自动化程度越高（人为的监管越少），越可能陷入麻烦之中。机器人的实现者需要做一些取舍——这些试探方式有助于避免问题的出现，但你可能会终止扫描那些看起来可疑的有效内容，因此这种方式也是“有损失”的。 222

在机器人会遇到的各种危险的 Web 中，有些技术的使用可以使机器人有更好的表现。

注 5：这是 http://www.searchtools.com/robots/robot-checklist.html 上提到的日历站点 http://cgi.umbc.edu/cgi-bin/WebEvent/Webevent.cgi 上的真实例子。这样的动态内容带来的后果就是，很多机器人都拒绝爬行 URL 中包含子字符串“cgi”的页面。

- 规范化 URL

  将 URL 转换为标准形式以避免语法上的别名

- 广度优先的爬行

  每次爬虫都有大量潜在的 URL 要去爬行。以广度优先的方式来调度 URL 去访问 Web 站点，就可以将环路的影响最小化。即使碰到了机器人陷阱，也可以在回到环路中获取的下一个页面之前，从其他 Web 站点中获取成百上千的页面。如果采用深度优先方式，一头扎到单个站点中去，就可能会跳入环路，永远无法访问其他站点。[6]

- 节流 [7]

  限制一段时间内机器人可以从一个 Web 站点获取的页面数量。如果机器人跳进了一个环路，试图不断地访问某个站点的别名，也可以通过节流来限制重复的页面总数和对服务器的访问总数。

- 限制 URL 的大小

  机器人可能会拒绝爬行超出特定长度（通常是 1KB）的 URL。如果环路使 URL 的长度增加，长度限制就会最终终止这个环路。有些 Web 服务器在使用长 URL 时会失败，因此，被 URL 增长环路困住的机器人会使某些 Web 服务器崩溃。这会让网管错误地将机器人当成发起拒绝服务攻击的攻击者。

  要小心，这种技术肯定会让你错过一些内容。现在很多站点都会用 URL 来管理用户的状态（比如，在一个页面引用的 URL 中存储用户 ID）。用 URL 长度来限制爬虫可能会带来些麻烦；但如果每当请求的 URL 达到某个特定长度时，都记录一次错误的话，就可以为用户提供一种检查某特定站点上所发生情况的方法。

- URL/ 站点黑名单

  维护一个与机器人环路和陷阱相对应的已知站点及 URL 列表，然后像躲避瘟疫一样避开它们。发现新问题时，就将其加入黑名单。

  这就要求有人工进行干预。但现在很多大型爬虫产品都有某种形式的黑名单，用于避开某些存在固有问题或者有恶意的站点。还可以用黑名单来避开那些对爬行

223 大惊小怪的站点。[8]

注 6：总之，广度优先搜索是个好方法，这样可以更均匀地分配请求，而不是都压到任意一台服务器上去。这样可以帮助机器人将用于一台服务器的资源保持在最低水平。

注 7：在 9.5 节也讨论了请求率的节流问题。

注 8：9.4 节讨论了站点怎样才能避免被爬行，但有些用户拒绝使用这种简单的控制机制，在其站点被爬行时又会变得非常愤怒。

- *模式检测*

  文件系统的符号连接和类似的错误配置所造成的环路会遵循某种模式；比如，URL 会随着组件的复制逐渐增加。有些机器人会将具有重复组件的 URL 当作潜在的环路，拒绝爬行带有多于两或三个重复组件的 URL。

  重复并不都是立即出现的（比如，“/subdir/subdir/subdir...”）。有些环路周期可能为 2 或其他间隔，比如“/subdir/images/subdir/images/subdir/images/...”。有些机器人会查找具有几种不同周期的重复模式。

- *内容指纹*

  一些更复杂的 Web 爬虫会使用指纹这种更直接的方式来检测重复。使用内容指纹的机器人会获取页面内容中的字节，并计算出一个*校验和*（checksum）。这个校验和是页面内容的压缩表示形式。如果机器人获取了一个页面，而此页面的校验和它曾经见过，它就不会再去爬行这个页面的链接了——如果机器人以前见过页面的内容，它就已经爬行过页面上的链接了。

  必须对校验和函数进行选择，以求两个不同页面拥有相同校验和的几率非常低。MD5 这样的报文摘要函数就常被用于指纹计算。

  有些 Web 服务器会在传输过程中对页面进行动态的修改，所以有时机器人会在校验和的计算中忽略 Web 页面内容中的某些部分，比如那些嵌入的链接。而且，无论定制了什么页面内容的动态服务器端包含（比如添加日期、访问计数等）都可能会阻碍重复检测。

- *人工监视*

  Web 就是一片荒野。勇敢的机器人最终总会陷入一个采用任何技术都无能为力的困境。设计所有产品级机器人时都要有诊断和日志功能，这样人类才能很方便地监视机器人的进展，如果发生了什么不寻常的事情就可以很快收到警告。在某些情况下，愤怒的网民会给你发送一些无礼的邮件来提示你出了问题。

爬行 Web 这样规模庞大的数据集时，好的蜘蛛探测法总是会不断改进其工作的。随着新的资源类型不断加入 Web，它会随着时间的推移构建出一些新的规则，并采纳这些规则。好的规则总是在不断发展之中的。

当受到错误爬虫影响的资源（服务器、网络带宽等）处于可管理状态，或者处于执行爬行工作的人的控制之下（比如在内部站点上）时，很多较小的、更具个性的爬虫就会绕开这些问题。这些爬虫更多的是依赖人类的监视来防止这些问题的发生。 224

9.2　机器人的HTTP

机器人和所有其他 HTTP 客户端程序并没有什么区别。它们也要遵守 HTTP 规范中的规则。发出 HTTP 请求并将自己广播成 HTTP/1.1 客户端的机器人也要使用正确的 HTTP 请求首部。

很多机器人都试图只实现请求它们所查找内容所需的最小 HTTP 集。这会引发一些问题；但短期内这种行为不会发生什么改变。结果就是，很多机器人发出的都是 HTTP/1.0 请求，因为这个协议的要求很少。

9.2.1　识别请求首部

尽管机器人倾向于只支持最小的 HTTP 集，但大部分机器人确实实现并发送了一些识别首部——最值得一提的就是 `User-Agent` 首部。建议机器人实现者们发送一些基本的首部信息，以通知各站点机器人的能力、机器人的标识符，以及它是从何处起源的。

在追踪错误爬虫的所有者，以及向服务器提供机器人所能处理的内容类型时，这些信息都是很有用的。鼓励机器人实现者们使用的基本识别首部包括如下内容。

- `User-Agent`
  将发起请求的机器人名字告知服务器。
- `From`
  提供机器人的用户 / 管理者的 E-mail 地址。[9]
- `Accept`
  告知服务器可以发送哪些媒体类型。[10] 这有助于确保机器人只接收它感兴趣的内容（文本、图片等）。
- `Referer`
  提供包含了当前请求 URL 的文档的 URL。[11]

9.2.2　虚拟主机

225 机器人实现者要支持 `Host` 首部。随着虚拟主机（参见第 5 章）的流行，请求中不

注 9：一种 RFC 822 E-mail 地址格式。

注 10：3.5.2 节列出了所有 `Accept` 相关的首部；机器人可能会发现，如果它们对特定版本感兴趣的话，发送 `Accept-Charset` 之类的首部是很有帮助的。

注 11：有些站点管理者会尝试着记录机器人是如何找到指向其站点内容的链接的，对这些人来说，这个首部非常有用。

包含 Host 首部的话，可能会使机器人将错误的内容与一个特定的 URL 关联起来。因此，HTTP/1.1 要求使用 Host 首部。

在默认情况下，大多数服务器都被配置为提供一个特定的站点。因此，不包含 Host 首部的爬虫向提供两个站点的服务器发起请求时，就像图 9-5 中的站点一样（www.joes-hardware.com 和 www.foo.com），假设默认情况下服务器被配置为提供 www.joes-hardware.com 站点（且不需要 Host 首部），那么，若请求 www.foo.com 上的某个页面，爬虫实际获取的就是 Joe 的五金商店的站点上的内容。更糟糕的是，爬虫会认为来自 Joe 的五金站点上的那些内容是来自 www.foo.com 的。如果带有相对立的政治色彩或其他观点的两个站点是由同一台服务器提供的，你肯定能想象到会有更不幸的局面出现。



图 9-5　发送请求时没有携带 **Host** 首部，虚拟 docroot 会引发问题的例子

9.2.3　条件请求

鉴于这些机器人的努力程度，尽量减少机器人所要获取内容的数量通常是很有意义的。对因特网搜索引擎机器人来说，需要下载的潜在页面有数十亿，所以，只在内容发生变化时才重新获取内容是很有意义的。

有些机器人实现了条件 HTTP 请求，[12] 它们会对时间戳或实体标签进行比较，看看它们最近获取的版本是否已经升级了。这与 HTTP 缓存查看已获取资源的本地副本是否有效的方法非常相似。更多与缓存对资源本地副本的验证有关的信息请参见第 7 章。 226

注 12：3.5.2 节给出了一个机器人可以实现的条件首部的完整列表。

9.2.4 对响应的处理

很多机器人的兴趣主要在于用简单的 GET 方法来获取所请求的内容，所以，一般不会在处理响应的方式上花费太多时间。但是，使用了某些 HTTP 特性（比如条件请求）的机器人，以及那些想要更好地探索服务器，并与服务器进行交互的机器人则要能够对各种不同类型的 HTTP 响应进行处理。

1. 状态码

总之，机器人至少应该能够处理一些常见的，以及预期的状态码。所有机器人都应该理解 200 OK 和 404 Not Found 这样的状态码。它们还应该能够根据响应的一般类别对它并不十分理解的状态码进行处理。第 3 章的表 3-2 给出了不同状态码的分类及其含义。

有些服务器并不总能返回适当的错误代码，认识到这一点是很重要的。有些服务器甚至会将 HTTP 状态码 200 OK 与描述错误状态的报文主体文本一同返回！很难对此做些什么——只是实现者应该要了解这些情况。

2. 实体

除了 HTTP 首部所嵌的信息之外，机器人也会在实体中查找信息。HTML 元标签，[13] 比如元标签 `http-equiv`，就是内容编写者用于嵌入资源附加信息的一种方式。

服务器可能会为它所处理的内容提供一些首部，标签 `http-equiv` 为内容编写者提供了一种覆盖这些首部的方式：

```
<meta http-equiv="Refresh" content="1;URL=index.html">
```

这个标签会指示接收者处理这个文档时，要当作其 HTTP 响应首部中有一个值为 `1,URL=index.html` 的 `Refresh` HTTP 首部。[14]

有些服务器实际上会在发送 HTML 页面之前先对其内容进行解析，并将 `http-equiv` 指令作为首部包含进去；有些服务器则不会。机器人实现者可能会去扫描 HTML 文
227 档的 HEAD 组件，以查找 `http-equiv` 信息。[15]

注 13：9.4.7 节列出了一些附加的元指令，站点管理员和内容编写者可以通过这些元指令来控制机器人的行为，以及这些机器人对已获取文档所执行的操作。

注 14：有时会将 `Refresh` HTTP 首部作为将用户（或者在这种情况下，就是将机器人）从一个页面重定向到另一个页面的手段。

注 15：根据 HTML 的规范，元标签一定要出现在 HTML 文档的 HEAD 部分。但并不是所有的 HTML 文档都会遵循规范，因此，它们有时也会出现在 HTML 文档的其他区域中。

9.2.5 User-Agent导向

Web 管理者应该记住，会有很多的机器人来访问它们的站点，因此要做好接收机器人请求的准备。很多站点会为不同的用户代理进行内容优化，并尝试着对浏览器类型进行检测，以确保能够支持各种站点特性。这样的话，当实际的 HTTP 客户端根本不是浏览器，而是机器人的时候，站点为机器人提供的就会是出错页面而不是页面内容了。在某些搜索引擎上执行文本搜索，搜索短语“your browser does not support frames”（你的浏览器不支持框架），会生成一个包含那条短语的出错页面列表。

站点管理者应该设计一个处理机器人请求的策略。比如，它们可以为所有其他特性不太丰富的浏览器和机器人开发一些页面，而不是将其内容限定在特定浏览器所支持的范围。至少，管理者应该知道机器人是会访问其站点的，不应该在机器人访问时感到猝不及防。[16]

9.3 行为不当的机器人

不守规矩的机器人会造成很多严重问题。这里列出了一些机器人可能会犯的错误，及其恶劣行为所带来的后果。

- 失控机器人

  机器人发起 HTTP 请求的速度要比在 Web 上冲浪的人类快得多，它们通常都运行在具有快速网络链路的高速计算机上。如果机器人存在编程逻辑错误，或者陷入了环路之中，就可能会向 Web 服务器发出大量的负载——很可能会使服务器过载，并拒绝为任何其他人提供服务。所有的机器人编写者都必须特别小心地设计一些保护措施，以避免失控机器人带来的危害。

- 失效的 URL

  有些机器人会去访问 URL 列表。这些列表可能很老了。如果一个 Web 站点对其内容进行了大量的修改，机器人可能会对大量不存在的 URL 发起请求。这会激怒某些 Web 站点的管理员，他们不喜欢他们的错误日志中充满了对不存在文档的访问请求，也不希望提供出错页面的开销降低其 Web 服务器的处理能力。

- 很长的错误 URL

  由于环路和编程错误的存在，机器人可能会向 Web 站点请求一些很大的、无意义的 URL。如果 URL 足够长的话，就会降低 Web 服务器的性能，使 Web 服务

注 16：如果某站点上有一些不应该让机器人访问的内容，站点管理员该如何控制机器人在其站点上的行为呢？9.4 节给出了相关的信息。

228 器的访问日志杂乱不堪，甚至会使一些比较脆弱的 Web 服务器崩溃。

- *爱打听的机器人*

  有些机器人可能会得到一些指向私有数据的 URL，这样，通过因特网搜索引擎和其他应用程序就可以很方便地访问这些数据了。如果数据的所有者没有主动宣传这些 Web 页面，那么在最好的情况下，他只是会认为机器人的发布行为惹人讨厌，而在最坏的情况下，则会认为这种行为是对隐私的侵犯。[17]

  通常，发生这种情况是由于机器人所跟踪的、指向"私有"内容的超链已经存在了（也就是说，这些内容并不像其所有者认为的那么隐密，或者其所有者忘记删除先前存在的超链了）。偶尔也会因为机器人非常热衷于寻找某站点上的文档而出现这种情况，很可能就是在没有显式超链的情况下去获取某个目录的内容造成的。

  从 Web 上获取大量数据的机器人的实现者们应该清楚，他们的机器人很可能会在某些地方获得敏感的数据——站点的实现者不希望通过因特网能够访问到这些数据。这些敏感数据可能包含密码文件，甚至是信用卡信息。很显然，一旦被指出，就应该有某种机制可以将这些数据丢弃（并从所有搜索索引或归档文件中将其删除），这是非常重要的。现在已知一些恶意使用搜索引擎和归档的用户会利用大型 Web 爬虫来查找内容——有些搜索引擎，比如 Google，[18] 实际上会对它们爬行过的页面进行归档，这样，即使内容被删除了，在一段时间内还是可以找到并访问它。

- *动态网关访问*

  机器人并不总是知道它们访问的是什么内容。机器人可能会获取一个内容来自网关应用程序的 URL。在这种情况下，获取的数据可能会有特殊的目的，计算的开销可能很高。很多 Web 站点管理员并不喜欢那些去请求网关文档的幼稚机器人。

9.4　拒绝机器人访问

机器人社团能够理解机器人访问 Web 站点时可能引发的问题。1994 年，人们提出了一项简单的自愿约束技术，可以将机器人阻挡在不适合它的地方之外，并为网站

注 17：通常，如果某资源可以通过公共因特网获取的话，它很可能会在某处被引用。由于因特网上链路网的存在，很少有资源是真正私有的。

注 18：参见 http://www.google.com 上的搜索结果。已缓存链接就是 Google 爬虫解析并索引过的页面的副本，大多数搜索结果中都会有已缓存链接。

管理员提供了一种能够更好地控制机器人行为的机制。这个标准被称为“拒绝机器人访问标准”，但通常只是根据存储访问控制信息的文件而将其称为 robots.txt。

robots.txt 的思想很简单。所有 Web 服务器都可以在服务器的文档根目录中提供一
个可选的、名为 robots.txt 的文件。这个文件包含的信息说明了机器人可以访问服务
器的哪些部分。如果机器人遵循这个自愿约束标准，它会在访问那个站点的所有其 229
他资源之前，从 Web 站点请求 robots.txt 文件。例如，图 9-6 中的机器人想要从 Joe
的五金商店下载 http://www.joes-hardware.com/specials/acetylene-torches.html。但在
机器人去请求这个页面之前，要先去查看 robots.txt 文件，看看它是否有获取这个页
面的权限。在这个例子中，robots.txt 文件并没有拦截机器人，因此机器人获取了这
个页面。

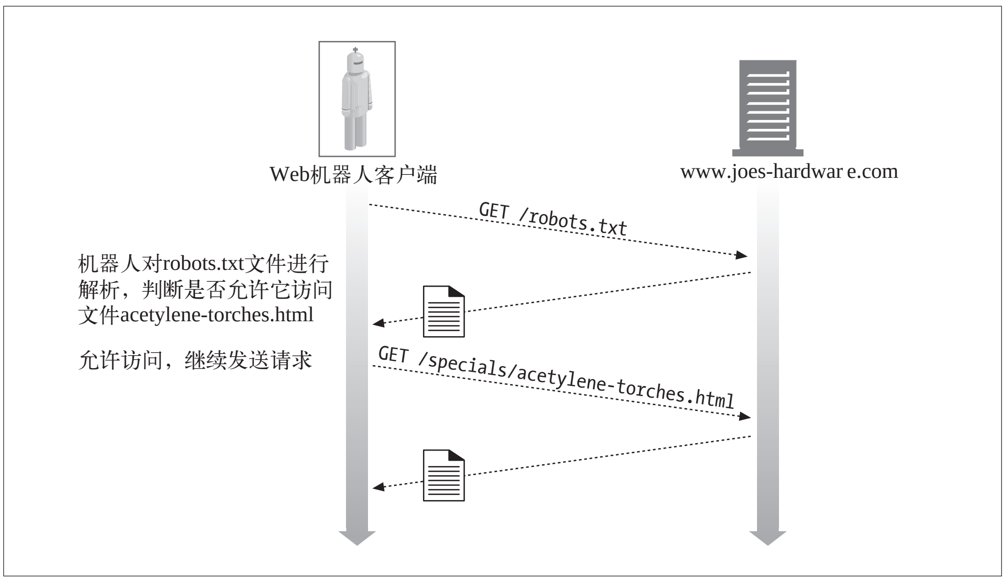


图 9-6　在爬行目标文件之前，先获取 robots.txt，验证是否可以进行访问

9.4.1　拒绝机器人访问标准

拒绝机器人访问标准是一个临时标准。编写本书的时候还没有官方标准机构承认这个标准，不同的厂商实现了这个标准的不同子集。但是，具备一些对机器人访问 Web 站点的管理能力，即使并不完美，也总比一点儿都没有要好，而且大部分主要的生产厂商和搜索引擎爬虫都支持这个拒绝访问标准。

尽管没有很好地定义版本的名称，但拒绝机器人访问标准是有三个版本的。我们采用了表 9-2 列出的版本编号。

表9-2　拒绝机器人访问标准的版本

版　本	标题及描述	日　期
0.0	拒绝机器人标准——Martijn Koster 提出的带有 `Disallow`（不允许）指令的原始 robots.txt 机制	1994 年 6 月
1.0	控制 Web 机器人的方法——Martijn Koster 提供了额外支持 `Allow`（允许）的 IETF 草案	1996 年 11 月
2.0	拒绝机器人访问的扩展标准——Sean Conner 提出的扩展标准，包括了正则表达式和定时信息；没有得到广泛的支持	1996 年 11 月

230

现在大多数机器人采用的都是标准 v0.0 或 v1.0。版本 v2.0 要复杂得多，没有得到广泛的应用。可能永远也不会得到广泛应用。这里我们重点介绍 v1.0 标准，因为它的应用很广泛，而且与 v0.0 完全兼容。

9.4.2　Web站点和robots.txt文件

如果一个 Web 站点有 robots.txt 文件，那么在访问这个 Web 站点上的任意 URL 之前，机器人都必须获取它并对其进行处理。[19] 由主机名和端口号定义的整个 Web 站点上仅有一个 robots.txt 资源。如果这个站点是虚拟主机，每个虚拟的 docroot 都可以有一个不同的 robots.txt 文件，像所有其他文件一样。

通常不能在 Web 站点上单独的子目录中安装“本地”robots.txt 文件。网管要负责创建一个聚合型 robots.txt 文件，用以描述 Web 站点上所有内容的拒绝访问规则。

1. 获取robots.txt

机器人会用 HTTP 的 GET 方法来获取 robots.txt 资源，就像获取 Web 服务器上所有其他资源一样。如果有 robots.txt 文件的话，服务器会将其放在一个 text/plain 主体中返回。如果服务器以 404 Not Found HTTP 状态码进行响应，机器人就可以认为这个服务器上没有机器人访问限制，它可以请求任意的文件。

机器人应该在 `From` 首部和 `User-Agent` 首部中传输标识信息，以帮助站点管理者对机器人的访问进行跟踪，并在站点管理者要查询，或投诉的机器人事件中提供一些联系信息。下面是一个来自商业 Web 机器人的 HTTP 爬虫请求实例：

```
GET /robots.txt HTTP/1.0
Host: www.joes-hardware.com
User-Agent: Slurp/2.0
Date: Wed Oct 3 20:22:48 EST 2001
```

注 19：尽管我们说的是 robots.txt 文件，但 robots.txt 资源并不一定要严格地位于文件系统中。比如，可以由一个网关应用程序动态地生成这个 robots.txt 资源。

2. 响应码

很多 Web 站点都没有 robots.txt 资源，但机器人并不知道这一点。它必须尝试着从每个站点上获取 robots.txt 资源。机器人会根据对 robots.txt 检索的结果采取不同的行动。

- 如果服务器以一个成功状态（HTTP 状态码 2XX）为响应，机器人就必须对内容进行解析，并使用排斥规则从那个站点上获取内容。
- 如果服务器响应说明资源并不存在（HTTP 状态码 404），机器人就可以认为服务器没有激活任何排斥规则，对此站点的访问不受 robots.txt 的限制。 231
- 如果服务器响应说明有访问限制（HTTP 状态码 401 或 403），机器人就应该认为对此站点的访问是完全受限的。
- 如果请求尝试的结果是临时故障（HTTP 状态码 503），机器人就应该推迟对此站点的访问，直到可以获取该资源为止。
- 如果服务器响应说明是重定向（HTTP 状态码 3XX），机器人就应该跟着重定向，直到找到资源为止。

9.4.3 robots.txt文件的格式

robots.txt 文件采用了非常简单的，面向行的语法。robots.txt 文件中有三种类型的行：空行、注释行和规则行。规则行看起来就像 HTTP 首部（<Field>:<value>）一样，用于模式匹配。比如：

```
# this robots.txt file allows Slurp & Webcrawler to crawl
# the public parts of our site, but no other robots...

User-Agent: slurp
User-Agent: webcrawler
Disallow: /private

User-Agent: *
Disallow:
```

robots.txt 文件中的行可以从逻辑上划分成“记录”。每条记录都为一组特定的机器人描述了一组排斥规则。通过这种方式，可以为不同的机器人使用不同的排斥规则。

每条记录中都包含了一组规则行，由一个空行或文件结束符终止。记录以一个或多个 `User-Agent` 行开始，说明哪些机器人会受此记录的影响，后面跟着一些 `Disallow` 和 `Allow` 行，用来说明这些机器人可以访问哪些 URL。[20]

注 20：出于实际应用的原因，机器人软件应该很强壮，可以灵活地使用行结束符。应该支持 CR、LF 和 CRLF。

前面的例子显示了一个 robots.txt 文件，这个文件允许机器人 Slurp 和 Webcrawler 访问除了 private 子目录下那些文件之外所有的文件。这个文件还会阻止所有其他机器人访问那个站点上的任何内容。

我们来看看 `User-Agent`、`Disallow` 和 `Allow` 行。

1. `User-Agent`行

每个机器人记录都以一个或多个下列形式的 `User-Agent` 行开始：

```
User-Agent: <robot-name>
```

或

232

```
User-Agent: *
```

在机器人 HTTP GET 请求的 `User-Agent` 首部中发送（由机器人实现者选择的）机器人名。

机器人处理 robots.txt 文件时，它所遵循的记录必须符合下列规则之一：

- 第一个 `<robot-name>` 是机器人名的大小写无关的子字符串；
- 第一个 `<robot-name>` 为“*”。

如果机器人无法找到与其名字相匹配的 `User-Agent` 行，而且也无法找到通配的 `User-Agent:*` 行，就是没有记录与之匹配，访问不受限。

由于机器人名是与大小写无关的子字符串进行匹配，所以要小心不要匹配错了。比如，`User-Agent:bot` 就与名为 Bot、Robot、Bottom-Feeder、Spambot 和 Dont-Bother-Me 的所有机器人相匹配。

2. `Disallow`和`Allow`行

`Disallow` 和 `Allow` 行紧跟在机器人排斥记录的 `User-Agent` 行之后。用来说明显式禁止或显式允许特定机器人使用哪些 URL 路径。

机器人必须将期望访问的 URL 按序与排斥记录中所有的 `Disallow` 和 `Allow` 规则进行匹配。使用找到的第一个匹配项。如果没有找到匹配项，就说明允许使用这个 URL。[21]

注 21：总是应该允许访问 robots.txt 的 URL，它一定不能出现在 `Allow/Disallow` 规则中。

要使 Allow/Disallow 行与一个 URL 相匹配，规则路径就必须是 URL 路径大小写相关的前缀。例如，Disallow: /tmp 就和下面所有的 URL 相匹配：

```
http://www.joes-hardware.com/tmp
http://www.joes-hardware.com/tmp/
http://www.joes-hardware.com/tmp/pliers.html
http://www.joes-hardware.com/tmpspc/stuff.txt
```

3. Disallow/Allow前缀匹配

下面是 Disallow/Allow 前缀匹配的一些细节。

- Disallow 和 Allow 规则要求大小写相关的前缀匹配。（与 User-Agent 行不同）这里的星号没什么特殊的含义，但空字符串可以起到通配符的效果。
- 在进行比较之前，要将规则路径或 URL 路径中所有“被转义”的字符（%XX）都反转为字节（除了正斜杠 %2F 之外，它必须严格匹配）。
- 如果规则路径为空字符串，就与所有内容都匹配。

表 9-3 列出了几个在规则路径和 URL 路径间进行匹配的例子。 233

表9-3　robots.txt路径匹配示例

规则路径	URL路径	匹配吗？	注　释
/tmp	/tmp	√	规则路径 == URL 路径
/tmp	/tmpfile.html	√	规则路径是 URL 路径的前缀
/tmp	/tmp/a.html	√	规则路径是 URL 路径的前缀
/tmp/	/tmp	×	/tmp/ 不是 /tmp 的前缀
	README.TXT	√	空的规则路径匹配于所有的路径
/~fred/hi.html	/%7Efred/hi.html	√	将 %7E 与 ~ 同等对待
/%7Efred/hi.html	/~fred/hi.html	√	将 %7E 与 ~ 同等对待
/%7efred/hi.html	/%7Efred/hi.html	√	转义符是大小写无关的
/~fred/hi.html	~fred%2Fhi.html	×	%2F 是一个斜杠，但斜杠是种特殊情况，必须完全匹配

前缀匹配通常都能很好地工作，但有几种情况下它的表达力却不够强。如果你希望无论使用什么路径前缀，都不允许爬行一些特别的子目录，那 robots.txt 是无能为力的。比如，你可能希望禁止在用于 RCS 版本控制的子目录中爬行。除了将到达各 RCS 子目录的每条路径都分别枚举出来之外，1.0 版的 robots.txt 方案无法提供此功能。

9.4.4 其他有关robots.txt的知识

解析 robots.txt 文件时还需遵循其他一些规则。

- 随着规范的发展，robots.txt 文件中可能会包含除了 `User-Agent`、`Disallow` 和 `Allow` 之外的其他字段。机器人应该将所有它不理解的字段都忽略掉。
- 为了实现后向兼容，不能在中间断行。
- 注释可以出现在文件的任何地方；注释包括可选的空格，以及后面的注释符（#）、注释符后面的注释，直到行结束符为止。
- 0.0 版的拒绝机器人访问标准并不支持 `Allow` 行。有些机器人只实现了 0.0 版的规范，因此会忽略 `Allow` 行。在这种情况下，机器人的行为会比较保守，有些允许访问的 URL 它也不去获取。

9.4.5 缓存和robots.txt的过期

如果一个机器人在每次访问文件之前都要重新获取 robots.txt 文件，Web 服务器上的负载就会加倍，机器人的效率也会降低。机器人使用的替代方法是，它会周期性地获取 robots.txt 文件，并将得到的文件缓存起来。机器人会使用这个 robots.txt 文
234 件的缓存副本，直到其过期为止。原始服务器和机器人都会使用标准的 HTTP 缓存控制机制来控制 robots.txt 文件的缓存。机器人应该留意 HTTP 响应中的 `Cache-Control` 和 `Expires` 首部。[22]

现在很多产品级爬虫都不是 HTTP/1.1 的客户端；网管应该意识到这些爬虫不一定能够理解那些为 robots.txt 资源提供的缓存指令。

如果没有提供 `Cache-Control` 指令，规范草案允许将其缓存 7 天。但实际上，这个时间通常太长了。不了解 robots.txt 文件的 Web 服务器管理员通常会在响应机器人的访问时创建一个新的文件，但如果将缺乏信息的 robots.txt 文件缓存一周，新创建的 robots.txt 文件就没什么效果了，站点管理员会责怪机器人管理员没有遵守拒绝机器人访问标准。[23]

9.4.6 拒绝机器人访问的Perl代码

有几个公共的 Perl 库可以用来与 robots.txt 文件进行交互。CPAN 公共 Perl 文档中的 `WWW::RobotsRules` 模块就是一个这样的例子。

注 22：更多有关缓存指令处理方面的内容请参见 7.8 节。

注 23：有几种大型的 Web 爬虫，如果它们在 Web 上勤奋爬行的话，每天都会重新获取 robots.txt。

将已解析的 robots.txt 文件保存在 `WWW::RobotRules` 对象中，这个对象提供了一些方法，可以用于查看是否禁止对某指定 URL 进行访问。同一个 `WWW::RobotRules` 可以用于解析多个 robots.txt 文件。

下面是 `WWW::RobotRules API` 的一些主要方法。

- 创建 `RobotRules` 对象

  `$rules = WWW::RobotRules->new($robot_name);`

- 装载 robots.txt 文件

  `$rules->parse($url, $content, $fresh_until);`

- 查看站点 URL 是否可获取

  `$can_fetch = $rules->allowed($url);`

下面这个短小的 Perl 程序说明了 `WWW::RobotRules` 的用法：

```
require WWW::RobotRules;

# Create the RobotRules object, naming the robot "SuperRobot"
my $robotsrules = new WWW::RobotRules 'SuperRobot/1.0';
use LWP::Simple qw(get);

# Get and parse the robots.txt file for Joe's Hardware, accumulating
# the rules
$url = "http://www.joes-hardware.com/robots.txt";
my $robots_txt = get $url;
$robotsrules->parse($url, $robots_txt);

# Get and parse the robots.txt file for Mary's Antiques, accumulating
# the rules
$url = "http://www.mary's antiques.com/robots.txt";
my $robots_txt = get $url;
$robotsrules->parse($url, $robots_txt);

# Now RobotRules contains the set of robot exclusion rules for several
# different sites. It keeps them all separate. Now we can use RobotRules
# to test if a robot is allowed to access various URLs.
if ($robotsrules->allowed($some_target_url))
{
    $c = get $url;
    ...
}
```

235

下面是 www.marys-antiques.com 的假想 robots.txt 文件：

```
#####################################################################
# This is the robots.txt file for Mary's Antiques web site
#####################################################################
```

```
# Keep Suzy's robot out of all the dynamic URLs because it doesn't
# understand them, and out of all the private data, except for the
# small section Mary has reserved on the site for Suzy.

User-Agent: Suzy-Spider
Disallow: /dynamic
Allow: /private/suzy-stuff
Disallow: /private

# The Furniture-Finder robot was specially designed to understand
# Mary's antique store's furniture inventory program, so let it
# crawl that resource, but keep it out of all the other dynamic
# resources and out of all the private data.

User-Agent: Furniture-Finder
Allow: /dynamic/check-inventory
Disallow: /dynamic
Disallow: /private

# Keep everyone else out of the dynamic gateways and private data.

User-Agent: *
Disallow: /dynamic
Disallow: /private
```

这个 robots.txt 文件中包含了一条机器人 SuzySpider 的记录，一条机器人 FurnitureFinder 的记录，以及一条用于所有其他机器人的默认记录。每条记录都对不同的机器人使用了一组不同的访问策略。

- SuzySpider 的排斥记录不允许机器人爬行以 /dynamic 开头的商店库存网关 URL，以及在为 Suzy 保留的区域之外的其他私有用户数据。
- 236 FurnitureFinder 机器人的记录允许机器人爬行家具库存网关 URL。这个机器人可能能够理解 Mary 的网关格式和规则。
- 其他机器人都不能访问所有的动态和私有 Web 页面，但它们可以爬行其余的 URL。

表 9-4 列出了几个机器人实例，这几个机器人具有不同的 Mary 古董网站访问权限。

表9-4　Mary古董网站的机器人访问权限

URL	SuzySpider	FurnitureFinder	NosyBot
http://www.marys-antiques.com/	√	√	√
http://www.marys-antiques.com/index.html	√	√	√

（续）

URL	SuzySpider	FurnitureFinder	NosyBot
http://www.marys-antiques.com/private/payroll.xls	×	×	×
http://www.marys-antiques.com/private/suzy-stuff/taxes.txt	√	×	×
http://www.marys-antiques.com/dynamic/buy-stuff?id=3546	×	×	×
http://www.marys-antiques.com/dynamic/check-inventory?kitchen	×	√	×

9.4.7 HTML的robot-control元标签

robots.txt 文件允许站点管理员将机器人排除在 Web 站点的部分或全部内容之外。robots.txt 文件的一个缺点就是它是 Web 站点管理员，而不是各部分内容的作者所有的。

HTML 页面的作者有一种更直接的方式可以限制机器人访问那些独立的页面。他们可以直接在 HMTL 文档中添加 robot-control 标签。遵循 robot-control HTML 标签规则的机器人仍然可以获取文档，但如果其中有机器人排斥标签，它们就会忽略这些文档。比如，因特网搜索引擎机器人就不会在其搜索索引中包含这个目录了。和 robots.txt 标准一样，鼓励但并不强制使用这个标签。

机器人排斥标签是以如下形式，通过 HTML 的 META 标签来实现的：

```
<META NAME="ROBOTS" CONTENT=directive-list>
```

1. 机器人的META指令

机器人 META 指令有几种不同的类型，而且随着时间的推移，以及搜索引擎及机器人对其行为和特性集的扩展，很可能还会添加一些新的指令。最常用的两个机器人 META 指令如下所示。

- NOINDEX 237

  告诉机器人不要对页面的内容进行处理，忽略文档（也就是说，不要在任何索引或数据库中包含此内容）。

```
<META NAME="ROBOTS" CONTENT="NOINDEX">
```

- NOFOLLOW

  告诉机器人不要爬行这个页面的任何外连链接。

  ```
  <META NAME="ROBOTS" CONTENT="NOFOLLOW">
  ```

除了 NOINDEX 和 NOFOLLOW 之外，还有相对应的 INDEX 指令、FOLLOW 指令、NOARCHIVE 指令以及 ALL 和 NONE 指令。下面对这些机器人 META 标签指令进行了总结。

- INDEX

  告诉机器人它可以对页面的内容进行索引。

- FOLLOW

  告诉机器人它可以爬行页面上的任何外连链接。

- NOARCHIVE

  告诉机器人不应该缓存这个页面的本地副本。[24]

- ALL

  等价于 INDEX、FOLLOW。

- NONE

  等价于 NOINDEX、NOFOLLOW。

与所有 HTML 的 META 标签类似，机器人 META 标签必须出现在 HTML 页面的 HEAD 区域中：

```
<html>
<head>
    <meta name="robots" content="noindex,nofollow">
    <title>...</title>
</head>
<body>
    ...
</body>
</html>
```

注意，标签的名称 robots 和内容都是大小写无关的。

很显然，不能发出一些会产生冲突或重复的指令，比如：

```
<meta name="robots" content="INDEX,NOINDEX,NOFOLLOW,FOLLOW,FOLLOW">
```

注 24：那些运行 Google 搜索引擎的人引入这个 META 标签，是为了向网管提供一种不允许 Google 提供其内容缓存页面的手段。此标签还可以与 META NAME="googlebot" 一起使用。

这种指令的行为很可能是未定义的，肯定会随机器人实现的不同而有所不同。

2. 搜索引擎的META标签

我们刚刚讨论了机器人的 META 标签，可以用来控制 Web 机器人的爬行和索引行为。所有的机器人 META 标签中都包含了 name="robots" 属性。 238

还有很多其他类型的 META 标签可用，包括表 9-5 所示的各种标签。对内容索引型搜索引擎机器人来说，DESCRIPTION 和 KEYWORDS META 标签都非常有用。

表9-5　其他META标签指令

name=	content=	描　述
DESCRIPTION	<文本>	允许作者为 Web 页面定义一个短小的文本摘要。很多搜索引擎都会查看 META DESCRIPTION 标签，允许页面作者指定一些短小的摘要来描述其 Web 页面 <meta name="description" content="Welcome to Mary's Antiques Web site">
KEYWORDS	<逗号列表>	关联一个由逗号分隔的 Web 页面描述词列表，为关键字搜索提供帮助 <meta name="keywords" content="antiques,mary,furniture,restoration">
REVISIT-AFTER[25]	<天数>	告诉机器人或搜索引擎应该在指定天数之后重访页面，估计那时候页面可能会发生变化 <meta name="revisit-after" content="10 days">

9.5　机器人的规范

1993 年，Web 机器人社会的先驱 Martijn Koster 为 Web 机器人的编写者们编写了一个指南列表。有些建议已经过时了，但有很多建议仍然非常有用。在 http://www.robotstxt.org/wc/guidelines.html 上可以找到 Martijn 的原始论文“Guidelines for Robot Writers”。

表 9-6 是为机器人设计者和操作人员提供的现代更新，这些更新的建议主要还是建立在原始列表的思想和内容之上的。大部分指南都是针对万维网机器人提出的；但它们同样适用于较小规模的爬虫。

注 25：这个指令很可能没有得到广泛的支持。

表9-6　Web机器人操作员指南

操作指南	描　　述
(1) 识别	
识别你的机器人	用 HTTP 的 `User-Agent` 字段将机器人的名字告诉 Web 服务器。这样可以帮助管理员理解机器人所做的事情。有些机器人还会在 `User-Agent` 首部包含一个描述机器人目的和策略的 URL
识别你的机器	确保机器人是从一台带有 DNS 条目的机器上运行的，这样 Web 站点才能够将机器人的 IP 地址反向 DNS 为主机名。这有助于管理者识别出对机器人负责的组织
239 识别联络人	用 HTTP 的 `From` 字段提供一个联络的 E-mail 地址
(2) 操作	
保持警惕	机器人可能会惹一些麻烦或引发一些抱怨。其中一些是由那些行为有偏差的机器人造成的。一定要小心，注意保持机器人的正常行为。如果机器人要全天候运行，就要格外小心。需要有操作人员不间断地对机器人进行监视，直到它有了丰富的经验为止
做好准备	开始一次重要的机器人之旅时，一定要通知你所在的组织。你的组织可能要观测网络带宽的耗费，作好应对各种公共查询的准备
监视并记录日志	机器人应该装备有丰富的诊断和日志记录工具，这样才能记录进展、识别所有的机器人陷阱，进行完整性检查看看工作是否正常。监视并记录机器人行为的重要性怎么强调也不过分。问题和抱怨总是会有的，对爬虫行为的详细记录，有助于机器人操作者回溯所发生的事情。不管是为了调试出错的 Web 爬虫，还是为了在不合理的投诉面前为其行为进行辩护，监视和记录工作都是非常重要的
学习并适应	在每次爬行中你都会学到新的东西。要让机器人逐步适应，这样，它在每次爬行之后都会有所进步，并能避开一些常见的陷阱
(3) 约束自己的行为	
对 URL 进行过滤	如果一个 URL 指向的好像是你不理解或不感兴趣的数据，你可能会希望跳过它。比如，以 .Z、.gz、.tar 或者 .zip 结尾的 URL 很可能是压缩文件或归档文件。以 .exe 结尾的 URL 可能就是程序。以 .gif、.tif、.jpg 结尾的 URL 很可能是图片。要确保你得到的就是你想要的
过滤动态 URL	通常，机器人不会想去爬行来自动态网关的内容。机器人不知道应该如何正确地格式化查询请求，并将其发送给网关，而它得到的结果也很可能是错误的或临时的。如果一个 URL 中包含了 cgi，或者有一个“?”，机器人可能就不会去爬行这个 URL 了
对 `Accept` 首部进行过滤	机器人应该用 HTTP 的 `Accept` 首部来告诉服务器它能够理解哪种内容
遵循 robots.txt	机器人应该接受站点上 robots.txt 的控制

（续）

操作指南	描　　述
制约自己	机器人应该记录访问每个站点的次数以及访问的时间，并通过这些信息来确保它没有太频繁地访问某个站点。机器人访问站点的频率高于几分钟一次时，管理员就要起疑心了。机器人每隔几秒钟就访问一次站点时，有些管理员就会生气了。机器人尽可能频繁地去访问一个站点，将所有其他流量都拒之门外时，管理员就会暴怒起来。 总之，应该限制机器人，使其每分钟最多只发送几条请求，并确保每条请求之间有几秒钟的间隔。还应该限制对站点的访问总次数，以防止环路的出现
(4) 容忍存在环路、重复和其他问题	
处理所有返回代码	必须做好处理所有 HTTP 状态码的准备，包括各种重定向和错误码。还应该对这些代码进行记录和监视。如果某站点出现大量不成功的结果，就应该对其进行调查。可能是很多 URL 过期了，或者服务器拒绝向机器人提供这些文档
规范 URL	试着将所有 URL 都转化为标准形式来消除常见的别名
积极地避免环路的出现	努力地检测并避免环路的出现。将操纵爬虫的过程当作一个反馈回路。应该将问题的结果和解决方法回馈到下一次爬行中，使爬虫在每次迭代之后都能表现得更好 240
监视陷阱	有些环路是故意造成的恶意环路。这些环路可能很难检测。有的站点会带有一些怪异的 URL，要监视对这类站点进行的大量访问。这种情况可能就是陷阱
维护一个黑名单	找到陷阱、环路、故障站点和不希望机器人访问的站点时，要将其加入一个黑名单，不要再次访问这些站点
(5) 可扩展性	
了解所需空间	事先通过数学计算明确你要解决的问题规模有多大。你可能会对应用程序完成一项机器人任务所需的内存规模感到非常吃惊，这是由 Web 庞大的规模造成的
了解所需带宽	了解你有多少网络带宽可用，以及在要求的时间内完成机器人任务所需的带宽大小。监视网络带宽的实际使用情况。你很可能会发现输出带宽（请求）要比输入带宽（响应）小得多。通过对网络使用情况的监视，可能还会找到一些方法来更好地优化你的机器人，通过更好地使用其 TCP 连接更好地利用网络带宽[26]
了解所需的时间	了解机器人完成其任务所需花费的时间，检查这个进度是否与自己的估计相符。如果机器人的耗费与自己的估计相去甚远，可能就会有问题，需要进行调查

注 26：更多有关 TCP 性能优化的内容请参见第 4 章。

（续）

操作指南	描　　述
分而治之	对大规模的爬行来说，很可能需要使用更多的硬件来完成这项工作，可以使用带有多个网卡的大型多处理器服务器，也可以使用多台较小的计算机共同配合工作
(6) 可靠性	
彻底测试	在将机器人放出去之前，要对其进行彻底的内部测试。作好非现场测试准备时，要先进行几次小型的处女航。收集大量结果并对性能和内存使用情况进行分析，估计一下它们会怎样累积成较大问题
检查点	所有严谨的机器人都要保存其进展的快照，出现故障时可以从那里重新开始。故障总是存在的：你会发现一些软件的 bug，硬件也会出故障。大规模机器人不能在每次出现这种情况时都从头开始。一开始就要设计检查点 / 重启机制
故障恢复	预测故障的发生，对机器人进行设计，使其能够在发生故障时继续工作
(7) 公共关系	
做好准备	机器人可能会让很多人感到困惑。要作好快速响应其询问的准备。制定一个 Web 页面政策声明，对机器人进行描述，其中包括创建 robots.txt 文件的详细指南
充分理解	有些与你联系，讨论机器人问题的人是了解情况并赞成的，有些人则很幼稚。少数人会异常愤怒。有些人看起来好像都要发疯了。去争辩机器人的努力有多么重要通常是没什么效果的。向他们解释拒绝机器人访问标准，如果他们仍然很不高兴，就立即将投诉者的 URL 从爬行列表中删除，并将其加入黑名单
积极响应	大多数不满意的网管都只是不太了解机器人。如果你能够进行迅速且专业的响应，90% 的投诉都会很快消失。另一方面，如果你等好几天才响应， 241 而机器人在继续访问这个站点，你面对的就将是一个非常愤怒的对手

9.6　搜索引擎

得到最广泛使用的 Web 机器人都是因特网搜索引擎。因特网搜索引擎可以帮助用户找到世界范围内涉及任意主题的文档。

现在 Web 上很多最流行的站点都是搜索引擎。很多 Web 用户将其作为起始点，它们会为用户提供宝贵的服务，帮助用户找到他们感兴趣的信息。

Web 爬虫为因特网搜索引擎提供信息，它们获取 Web 上的文档，并允许搜索引擎创建与本书后面的索引类似的索引，用以说明哪些文档中有哪些词存在。搜索引擎是 Web 机器人的主要来源——让我们来快速了解一下它们是如何工作的。

9.6.1　大格局

Web 发展的初期，搜索引擎就是一些相当简单的数据库，可以帮助用户在 Web 上定位文档。现在，Web 上有数十亿可供访问的页面，搜索引擎已经成为因特网用户查找信息不可缺少的工具。它们在不断地发展，以应对 Web 庞大的规模，因此，现在已经变得相当复杂了。

面对数十亿的 Web 页面，和数百万要查找信息的用户，搜索引擎要用复杂的爬虫来获取这数十亿 Web 页面，还要使用复杂的查询引擎来处理数百万用户产生的查询负荷。

我们来考虑一下产品级 Web 爬虫的任务，它要获取搜索索引所需的页面，它要发出数十亿条 HTTP 请求。如果每条请求都要花半秒钟的时间（对有些服务器来说可能慢了，对另一些服务器来说可能快了[27]），（对十亿份文件来说）就要花费：

0.5 秒 × （100 0000 000）/（60 秒 / 天）×（60 分 / 小时）×（24 小时 / 天）

如果请求是连续发出的，结果差不多是 5700 天！很显然，大型爬虫得更聪明一些，要对请求进行并行处理，并使用大量机器来完成这项任务。但由于其规模庞大，爬行整个 Web 仍然是件十分艰巨的任务。

9.6.2　现代搜索引擎结构

现在的搜索引擎都构建了一些名为“全文索引”的复杂本地数据库，装载了全世界的 Web 页面，以及这些页面所包含的内容。这些索引就像 Web 上所有文档的卡片目录一样。 242

搜索引擎爬虫会搜集 Web 页面，把它们带回家，并将其添加到全文索引中去。同时，搜索引擎用户会通过 HotBot（http://www.hotbot.com）或 Google（http://www.google.com）这样的 Web 搜索网关对全文索引进行查询。Web 页面总是在不断地发生变化，而且爬行一大块 Web 要花费很长的时间，所以全文索引充其量也就是 Web 的一个快照。

现代搜索引擎的高层结构如图 9-7 所示。

9.6.3　全文索引

全文索引就是一个数据库，给它一个单词，它可以立即提供包含那个单词的所有文档。创建了索引之后，就不需要对文档自身进行扫描了。

注 27：这取决于服务器的资源、客户端的机器人，以及两者之间的网络状况。

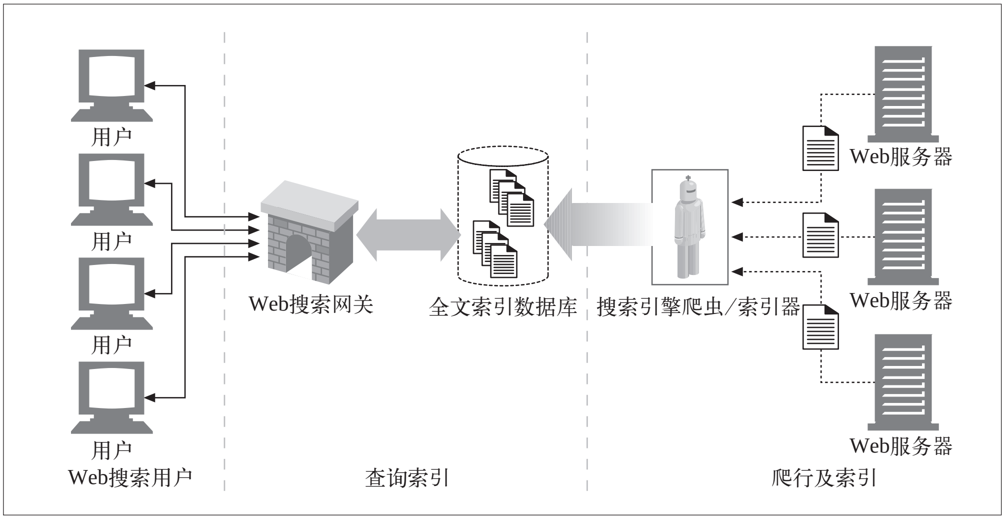


图 9-7　产品级搜索引擎中包含了一些协作的爬虫和查询网关

图 9-8 显示了三份文档和相应的全文索引。全文索引列出了包含每个单词的文档。

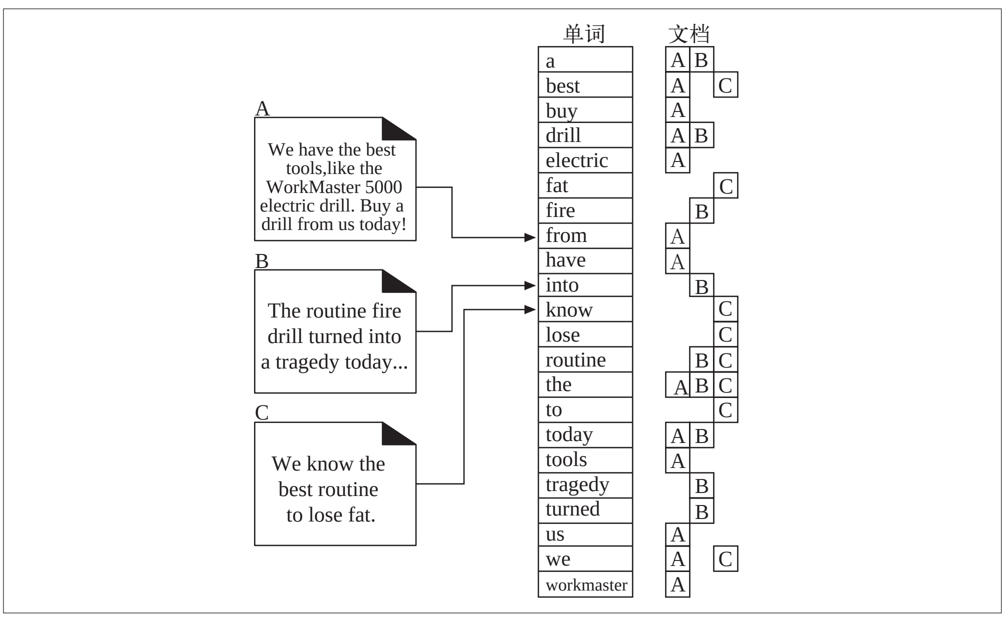


图 9-8　三份文档和一个全文索引

比如：

- 单词“a”位于文档 A 和 B 中；

- 单词“best”位于文档 A 和 C 中；
- 单词“drill”位于文档 A 和 B 中；
- 单词“routine”位于文档 B 和 C 中；
- 单词“the”位于所有的三份文档 A、B 和 C 中。 243

9.6.4 发布查询请求

用户向 Web 搜索引擎网关发布一条请求时，会填写一个 HTML 表单，他的浏览器会用一个 HTTP GET 或 POST 请求将这个表单发送给网关。网关程序对搜索请求进行解析，并将 Web UI 查询转换成搜索全文索引所需的表达式。[28]

图 9-9 显示了一条对 www.joes-hardware.com 站点的简单用户查询。用户在搜索框表单中输入 drills，然后浏览器就会将这个动作转换成一条在 URL 中包含请求参数的 GET 请求。[29]Joe 的五金商店的 Web 服务器收到这条请求，并将其转发给其搜索网关应用程序，这个程序会将文档的结果列表返回给 Web 服务器，然后 Web 服务器又会将这些结果转换成 HTML 页面提供给用户。

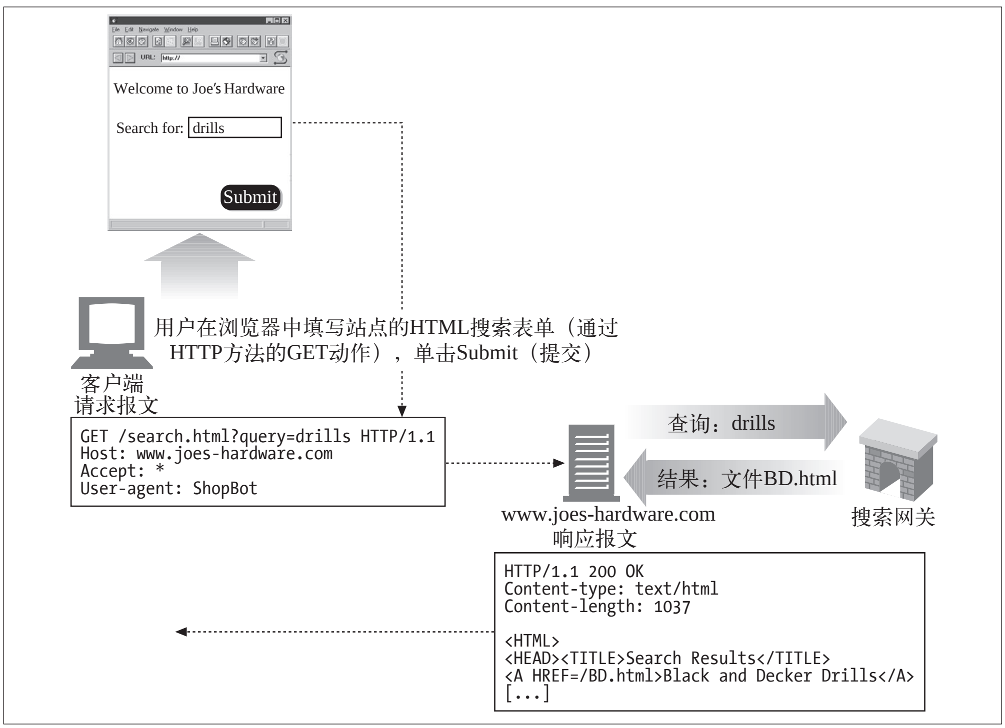


图 9-9 搜索查询请求的实例

注 28：传送这条请求的方法与所使用的搜索策略有关。

注 29：2.2.6 节讨论了 URL 中查询参数的常见用法。

9.6.5　对结果进行排序，并提供查询结果

一旦搜索引擎通过其索引得到了查询结果，网关应用程序会获取结果，并将其拼成
244 结果页面提供给终端用户。

很多 Web 页面都可能包含任意指定的单词，所以搜索引擎采用了一些很聪明的算法，尝试着对结果进行排名。比如，在图 9-8 中，单词 best 出现在很多文档中；为了将相关度最高的结果提供给用户，搜索引擎要知道应该按照什么顺序来提供结果列表中的文档。这被称为*相关性排名*（relevancy ranking）——这是对一系列搜索结果的评分和排序处理。

为了更好地辅助这一进程，在爬行 Web 的过程中都会进行数据统计。比如，对指向指定页面的链接进行计数有助于判断其流行程度，还可以用此信息来衡量提供结果的顺序。算法、爬行中获取的辅助信息以及搜索引擎所使用的其他技巧都是保守最森严的秘密。

9.6.6　欺诈

在搜索请求得到的前几个结果中没有看到自己想要查找的内容时，用户通常会感到很沮丧，因此，查找站点时搜索结果的顺序是很重要的。在搜索网管们认为能够最
245 好地描述其站点功能的单词时，会有众多因素激励着这些网管，努力使其站点排在靠近结果顶端的位置上，尤其是那些依赖于用户找到它们，并使用其服务的商业站点。

这种对较好排列位置的期待引发了很多对搜索系统的博弈，也在搜索引擎的实现者和那些想方设法要将其站点列在突出位置的人之间引发了持久的拉锯战。很多网管都列出了无数关键字（有些是毫不相关的），使用一些假冒页面，或者采用*欺诈*（spoof）行为——甚至会用网关应用程序来生成一些在某些特定单词上可以更好地欺骗搜索引擎相关性算法的假冒页面。

这么做的结果就是，搜索引擎和机器人实现者们要不断地修改相关性算法，以便更有效地抓住这些欺诈者。

9.7　更多信息

更多有关 Web 客户端的信息，请参见下列资料。

- http://www.robotstxt.org/wc/robots.html
  Web 机器人页面——机器人开发者所需的资源，包括因特网机器人的登记注册。

- http://www.searchengineworld.com
  搜索引擎世界——搜索引擎和机器人的相关资源。

- http://www.searchtools.com
  Web 站点和内部网络的搜索工具——搜索工具和机器人的相关资源。

- http://search.cpan.org/doc/ILYAZ/perl_ste/WWW/RobotRules.pm
  RobotRules 的 Perl 语言资源。

- http://www.conman.org/people/spc/robots2.html
  拒绝机器人访问的扩展标准。

- *Managing Gigabytes: Compressing and Indexing Documents and Images*（《海量数据管理——文档和图像的压缩与索引》）
  Witten, I.、Moffat, A. 和 Bell, T. 编写，Morgan Kaufmann 公司出版。 246

第10章

HTTP-NG

在这本书即将完稿的时候，HTTP 正在庆祝它的第十个生日。这十年是这个因特网协议成就辉煌的十年。现在，世界上绝大多数的数字流量都是由 HTTP 传输的。

但随着 HTTP 迈入青少年时期，它也面临着一些挑战。从某些方面来说，HTTP 应用的步伐已经超越了其设计。现在，人们将 HTTP 作为各种不同应用程序的基础，并将其运行在很多不同的联网技术之上。

本章对 HTTP 未来的一些发展趋势和所要面临的挑战，以及被称为 HTTP-NG 的下一代 HTTP 结构方案进行了介绍。尽管 HTTP-NG 工作组已经解散了，而且看起来也不太可能得到快速的应用，但它还是给出了 HTTP 未来一些潜在的发展方向。

10.1 HTTP发展中存在的问题

HTTP 最初被设想为一种简单的技术，用于访问分布式信息服务器上链接的多媒体内容。但在过去的十年中，HTTP 及其衍生产品起到了更为广泛的作用。

HTTP/1.1 现在提供了可以追踪文档版本的标记和指纹，提供了一些方法来支持文档的上传以及与可编程网关之间的交互，还提供对多语言内容、安全及认证功能、降低流量的缓存功能、减小时延的管道功能、降低启动时间提高带宽使用效率的持久连接，以及用来进行部分更新的访问范围功能的支持。HTTP 的扩展及衍生产品具有更为广泛的功能，可以提供对文档发布、应用程序服务、任意的消息服务、视频
247 流以及无线多媒体访问的支持。HTTP 正在成为分布式多媒体应用程序的“操作系统”。

尽管 HTTP/1.1 的设计经过了充分的考量，但随着 HTTP 被越来越多地用作复杂远程操作的统一载体，HTTP/1.1 已经开始显现出了一些局限性。HTTP 的发展中至少存在 4 个方面的问题。

- 复杂性

  HTTP 相当复杂，而且其特性之间是相互依存的。由于存在一些复杂的、相互交织的要求，以及连接管理、报文处理和功能逻辑之间的混合作用，要想正确地实现 HTTP 软件肯定是非常痛苦、很容易出错的。

- 可扩展性

  HTTP 很难实现递增式扩展。很多流传下来的 HTTP 应用程序中都没有自主的功能性扩展技术，使协议的扩展无法兼容。

- 性能

  HTTP 中有些部分效率不高。其中很多低效特性会随着高时延、低吞吐量的无线

访问技术的广泛使用而变得更加严重。

- 传输依赖性

  HTTP 是围绕 TCP/IP 网络协议栈设计的。尽管没有限制说不能使用替代协议栈，但在这方面所做的工作非常少。HTTP 要为替代协议栈提供更多的支持，才能作为一个更广阔的报文发送平台应用于嵌入式和无线应用程序之中。

10.2 HTTP-NG的活动

1997 年夏天，万维网联盟启动了一个特殊项目，调查并提出一个新的 HTTP 版本，以修正与复杂性、可扩展性、性能及传输依赖性有关的一些问题。这个新的 HTTP 被称为 HTTP：下一代系统（HTTP-NG）。

在 1998 年 12 月举行的一次 IETF 会议上，提出了一组 HTTP-NG 建议。这些建议勾勒出了一种可能的 HTTP 主要发展方向。这项技术还未被广泛采用（可能永远也不会被广泛采用了），但 HTTP-NG 确实在扩展 HTTP 系统方面做出了最认真的努力。下面我们来仔细看看 HTTP-NG。

10.3 模块化及功能增强

可以用三个英语单词来描述 HTTP-NG 的主题："模块化及功能增强"（modularize and enhance）。如图 10-1 所示，HTTP-NG 工作组建议将协议模块化为三层，而不是将连接管理、报文处理、服务器处理逻辑和协议方法全都混在一起。 248

- 第一层，报文传输层（message transport layer），这一层不考虑报文的功能，而是致力于端点间报文的不透明传输。报文传输层支持各种子协议栈（比如无线环境下的协议栈），主要负责处理高效报文传输及处理方面的问题。HTTP-NG 项目组为本层提出了一个名为 WebMUX 的协议。
- 第二层，远程调用层（remote invocation layer），定义了请求 / 响应的功能，客户端可以通过这些功能调用对服务器资源的操作。本层独立于报文的传输以及操作的精确语义。它只是提供了一种标准的方法来调用服务器上所有的操作。本层试图提供一种像 CORBA、DCOM 和 Jave RMI 那样的面向对象的可扩展框架，而不是 HTTP/1.1 中那种静态的、服务器端定义的方法。HTTP-NG 项目组建议本层使用二进制连接协议（Binary Wire Protocol）。
- 第三层，Web 应用层（Web application layer），提供了大部分的内容管理逻辑。所有的 HTTP/1.1 方法（GET、POST、PUT 等），以及 HTTP/1.1 首部参数都是在这里定义的。本层还支持其他构建在远程调用基础上的服务，比如 WebDAV。



图 10-1　HTTP-NG 将功能都分散到各层之中实现

只要将 HTTP 组件模块化了，就可以对其进行改进，以提供更好的性能和更丰富的特性。

10.4　分布式对象

HTTP-NG 的很多基本原理和功能目标都是从 CORBA 和 DCOM 这样的结构化、面向对象的分布式对象系统中借鉴来的。分布式对象系统对可扩展性和功能特性都很有帮助。

从 1996 年开始，一个研究团体就在争论是否要将 HTTP 与更复杂的分布式对象系统聚合在一起。在 Web 中使用分布式对象模型有很多好处，更多与此相关的信息请查阅 Xerox PARC 早期名为“Migrating the Web Toward Distributed Objects”（“Web
249 向分布式对象的迁移”）的文章（ftp://ftp.parc.xerox.com/pub/ilu/misc/Webilu.html）。

将 Web 和分布式对象统一起来的雄心使得 HTTP-NG 的应用受到了某些社团的抵制。过去的一些分布式对象系统受到了重量级实现方案和形式上复杂性的影响。HTTP-NG 项目组也尝试去解决需求中提到的一些问题。

10.5　第一层——报文传输

我们从最底层开始，近距离地看看 HTTP-NG 这三层的功能。报文传输层关心的是报文的有效传输，不考虑报文的含义和目的。报文传输层为报文传输提供了一个 API，无论底层实际采用的是什么网络协议栈都可以使用。

本层关注的是提高报文传输的性能，其中包括：

- 对报文进行管道化和批量化传输，以降低往返时延；
- 重用连接，以降低时延，提高传输带宽；
- 在同一条连接上并行地复用多个报文流，在防止报文流饿死的同时优化共享连接；
- 对报文进行有效的分段，使报文边界的确定更加容易。

HTTP-NG 工作组将大部分精力都放在了为第一层的报文传输开发 WebMUX 协议上。WebMUX 是个高性能的报文协议，可以对报文进行分段，并在一条复用的 TCP 连接上交错地传输报文。本章会对 WebMUX 进行较为详细的介绍。

10.6 第二层——远程调用

HTTP-NG 结构的中间层提供了对远程方法调用的支持。本层提供了通用的请求 / 响应框架，客户端可通过此框架调用对服务器资源的操作。本层并不关心特定操作的实现及语义（缓存、安全性以及方法逻辑等）；它只关心允许客户端远程调用服务器操作的接口。

现在已经有很多远程方法调用标准了（举几个例子来说，比如 CORBA、DOM 和 Java RMI），本层并不打算支持这些系统中所有好的特性。但它有一个明确的目标，就是要对 HTTP/1.1 所提供的 HTTP RMI 支持进行扩展。特别是，要以可扩展的面向对象方式提供更通用的远程过程调用支持。

HTTP-NG 小组建议本层采用二进制连接协议。这个协议支持一种高性能的可扩展技术，通过这种技术可以调用服务器上经过良好描述的操作，并将结果返回。本章稍后将对二进制连接协议进行较为详细的讨论。 250

10.7 第三层——Web应用

Web 应用层是执行语义和应用程序特定逻辑的地方。HTTP-NG 工作组避开了扩展 HTTP 应用特性的诱惑，专注于正规的基础建设工作。

Web 应用层描述了一个用于提供应用程序特定服务的系统。这些服务并不单一，不同的应用程序可能使用不同的 API。比如，HTTP/1.1 的 Web 应用构成的应用程序与 WebDAV 可能会共享一些公用的部分，但又会有所不同。HTTP-NG 结构允许多个应用共存于本层，共享底层特性，它还提供了一种添加新应用程序的机制。

Web 应用层的基本思想是提供与 HTTP/1.1 等价的功能和一些扩展接口，同时将其映射到一个可扩展的分布式对象框架中去。更多与 Web 应用层接口有关的内容可以参见 http://www.w3.org/Protocols/HTTP-NG/1998/08/draft-larner-nginterfaces-00.txt。

10.8 WebMUX

HTTP-NG 工作组花费了很多精力，为报文传输开发了 WebMUX 标准。WebMUX 是一个复杂的高性能报文系统，通过这个系统，可以在一个复用的 TCP 连接上并行

地传输报文。可以对以不同速度产生和消耗的独立报文流进行高效的分组，并将其复用到一条或少数几条 TCP 连接上去（参见图 10-2）。

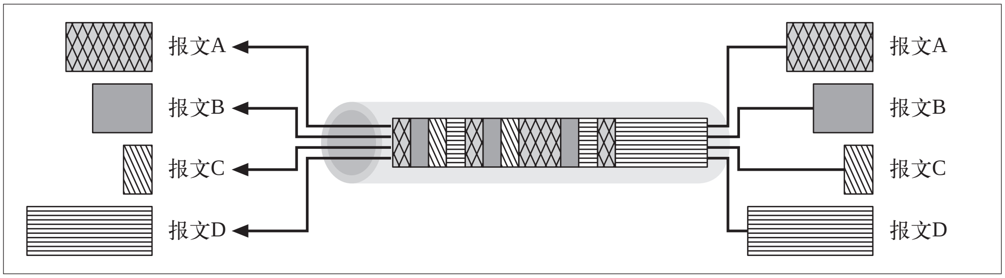


图 10-2　WebMUX 可以在一条连接上复用多条报文

WebMUX 协议的关键目标包括如下几条。

- 设计简单。
- 高性能。
- 复用——可以在一条连接上动态、高效地交错传递多个（使用任意高层协议的）
251 数据流，不用因为等待那些速度很慢的生产者程序而延迟数据的传输。
- 基于信用的流量控制——数据是以不同的速率产生和消耗的，发送者和接收者的内存和可用的 CPU 资源都有所不同。WebMUX 使用的是“基于信用的”流量控制方案，接收者可以预先声明期望的数据接收速度，防止出现资源缺乏产生的死锁。
- 保持对齐——保持复用流中数据的对齐，这样才能有效地发送并处理二进制数据。
- 丰富的功能——接口足够丰富，能支持套接字 API。

更多有关 WebMUX 协议的内容请参阅 http://www.w3.org/Protocols/MUX/WD-mux-980722.html。

10.9　二进制连接协议

HTTP-NG 小组建议使用二进制连接协议来提高下一代 HTTP 协议支持远程操作的能力。

HTTP-NG 定义了一些“对象类型”，并为每种对象类型分配了一组方法。为每种对象类型分配一个 URI，以便将对它的描述和它的方法宣传出去。通过这种方式，HTTP-NG 提供了一种比 HTTP/1.1 的扩展性更强，且面向对象的执行模型，HTTP/1.1 中所有的方法都是在服务器中静态定义的。

二进制连接协议通过一条有状态的连接承载了从客户端发往服务器的操作调用请求，以及从服务器发往客户端的操作结果应答。有状态的连接可以提供更高的效率。

请求报文中包含操作、目标对象和可选的数据值。应答报文带回了操作的最终状态、所对应请求的序列号（允许以任意顺序传递并行的请求和响应），以及可选的返回值。除了请求和应答报文之外，这个协议还定义了几种内部控制报文，用来提高连接的效率和强壮性。

更多有关二进制连接协议的内容请参阅 http://www.w3.org/Protocols/HTTP-NG/1998/08/draft-janssen-httpng-wire-00.txt。

10.10　当前的状态

1998 年底，HTTP-NG 小组认定现在将 HTTP-NG 建议引入 IETF 还为时尚早。工业界和各社团都还没有完全调整到 HTTP/1.1 上来，如果没有明确的转换计划
就进行重大的 HTTP-NG 重构计划，将其重构为分布式对象模式可能会造成严重 252
的破坏。

对此，人们提出了两点建议。

- 不要指望能在一步之内实现整个 HTTP-NG 重构计划，建议重点关注 WebMUX 传输技术。但是，在编写此书的时候，人们还没有足够的兴趣去建立一个 WebMUX 工作组。
- 要去探讨能否将正式的协议类型修改得足够灵活（可能是通过 XML 实现的），以满足 Web 上应用的需要。对于可扩展的分布式对象系统来说，这一点尤其重要。这项工作仍在进行之中。

编写此书时，还没有什么主导力量在驱动 HTTP-NG 的应用。但随着 HTTP 应用的不断增多，随着它越来越多地作为各种应用程序的平台使用，以及无线因特网技术和面向消费者的因特网技术的不断应用，HTTP-NG 中提出的一些技术可能会在 HTTP 的青少年时期逐渐显现出其重要性。

10.11　更多信息

更多有关 HTTP-NG 的信息，请参见下面列出的详细规范和活动报告。

- http://www.w3.org/Protocols/HTTP-NG/
  HTTP-NG 工作组（建议），W3C 联盟 Web 站点。

- http://www.w3.org/Protocols/MUX/WD-mux-980722.html
  J. Gettys 和 H. Nielsen 编写的“The WebMUX Protocol”（“WebMUX 协议”）。
- http://www.w3.org/Protocols/HTTP-NG/1998/08/draft-janssen-httpng-wire-00.txt
  B. Janssen 编写的“Binary Wire Protocol for HTTP-NG”（“HTTP-NG 的二进制连接协议”）。
- http://www.w3.org/Protocols/HTTP-NG/1998/08/draft-larner-nginterfaces-00.txt
  D. Larner 编写的“HTTP-NG Web Interfaces”（“HTTP-NG Web 接口”）。
- ftp://ftp.parc.xerox.com/pub/ilu/misc/Webilu.html
  D. Larner 编写的“Migrating the Web Toward Distributed Objects”（“Web 向分
  253 布式对象的迁移”）。

第三部分

识别、认证与安全

第三部分的 4 章提供了一系列的技术和技巧，可用来跟踪身份、进行安全性检查，控制对内容的访问。

- 第 11 章介绍了识别用户的技巧，这样就可以实现用户个性化的内容了。
- 第 12 章重点概述了用户身份验证的基本原理。这一章还探讨了 HTTP 认证与数据库之间的接口机制。
- 第 13 章解释了摘要验证，这是一种复杂的、针对 HTTP 的增强方式，可以极大地提高安全性。 254~256
- 第 14 章详细介绍了因特网密码学、数字证书和安全套接字层。

第11章

客户端识别与cookie机制

Web 服务器可能会同时与数千个不同的客户端进行对话。这些服务器通常要记录下它们在与谁交谈，而不会认为所有的请求都来自匿名的客户端。本章讨论了一些服务器可以用来识别其交谈对象的技巧。

11.1　个性化接触

HTTP 最初是一个匿名、无状态的请求 / 响应协议。服务器处理来自客户端的请求，然后向客户端回送一条响应。Web 服务器几乎没有什么信息可以用来判定是哪个用户发送的请求，也无法记录来访用户的请求序列。

现代的 Web 站点希望能够提供个性化的接触。它们希望对连接另一端的用户有更多的了解，并且能在用户浏览页面时对其进行跟踪。Amazon.com 这样流行的在线商店网站可以通过以下几种方式实现站点的个性化。

- *个性化的问候*
  专门为用户生成的欢迎词和页面内容，使购物体验更加个性化。

- *有的放矢的推荐*
  通过了解客户的兴趣，商店可以推荐一些它们认为客户会感兴趣的商品。商店还可以在临近客户生日或其他一些重要日子的时候提供生日特定的商品。

- *管理信息的存档*
  在线购物的用户不喜欢一次又一次地填写繁琐的地址和信用卡信息。有些站点会将这些管理细节存储在一个数据库中。只要他们识别出用户，就可以使用存档的
257 管理信息，使得购物体验更加便捷。

- *记录会话*
  HTTP 事务是无状态的。每条请求 / 响应都是独立进行的。很多 Web 站点希望能在用户与站点交互的过程中（比如，使用在线购物车的时候）构建增量状态。要实现这一功能，Web 站点就需要有一种方式来区分来自不同用户的 HTTP 事务。

本章对 HTTP 识别用户的几种技巧进行了总结。HTTP 并不是天生就具有丰富的识别特性的。早期的 Web 站点设计者们（他们都是些注重实际的人）都有自己的用户识别技术。每种技术都有其优势和劣势。本章我们将讨论下列用户识别机制。

- 承载用户身份信息的 HTTP 首部。
- 客户端 IP 地址跟踪，通过用户的 IP 地址对其进行识别。
- 用户登录，用认证方式来识别用户。
- 胖 URL，一种在 URL 中嵌入识别信息的技术。

• cookie，一种功能强大且高效的持久身份识别技术。

11.2 HTTP首部

表 11-1 给出了七种最常见的用来承载用户相关信息的 HTTP 请求首部。这里我们先讨论前三个；后面四个首部用于更高级的识别技术，我们稍后讨论。

表11-1 承载用户相关信息的HTTP首部

首部名称	首部类型	描 述
`From`	请求	用户的 E-mail 地址
`User-Agent`	请求	用户的浏览器软件
`Referer`	请求	用户是从这个页面上依照链接跳转过来的
`Authorization`	请求	用户名和密码（稍后讨论）
`Client-IP`	扩展（请求）	客户端的 IP 地址（稍后讨论）
`X-Forwarded-For`	扩展（请求）	客户端的 IP 地址（稍后讨论）
`Cookie`	扩展（请求）	服务器产生的 ID 标签（稍后讨论）

`From` 首部包含了用户的 E-mail 地址。每个用户都有不同的 E-mail 地址，所以在理想情况下，可以将这个地址作为可行的源端来识别用户。但由于担心那些不讲道德的服务器会搜集这些 E-mail 地址，用于垃圾邮件的散发，所以很少有浏览器会发送 `From` 首部。实际上，`From` 首部是由自动化的机器人或蜘蛛发送的，这样在出现问题时，网管还有个地方可以发送愤怒的投诉邮件。 258

`User-Agent` 首部可以将用户所用浏览器的相关信息告知服务器，包括程序的名称和版本，通常还包含操作系统的相关信息。要实现定制内容与特定的浏览器及其属性间的良好互操作时，这个首部是非常有用的，但它并没有为识别特定的用户提供太多有意义的帮助。下面是两种 `User-Agent` 首部，一种是网景的 Navigator 发送的，另一种是微软的 Internet Explorer 发送的：

• Navigator 6.2

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:0.9.4)
Gecko/20011128
  Netscape6/6.2.1
```

• Internet Explorer 6.01

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
```

`Referer` 首部提供了用户来源页面的 URL。`Referer` 首部自身并不能完全标识用户，但它确实说明了用户之前访问过哪个页面。通过它可以更好地理解用户的浏览

行为，以及用户的兴趣所在。比如，如果你是从一个篮球网站抵达某个 Web 服务器的，这个服务器可能会推断你是个篮球迷。

`From`、`User-Agent` 和 `Referer` 首部都不足以实现可靠的识别。后面的小节对识别特定用户的策略进行了更为详细的讨论。

11.3 客户端IP地址

早期的 Web 先锋曾尝试着将客户端 IP 地址作为一种标识形式使用。如果每个用户都有不同的 IP 地址，IP 地址（如果会发生变化的话）也很少会发生变化，而且 Web 服务器可以判断出每条请求的客户端 IP 地址的话，这种方案是可行的。通常在 HTTP 首部并不提供客户端的 IP 地址，[1] 但 Web 服务器可以找到承载 HTTP 请求的 TCP 连接另一端的 IP 地址。

比如，在 Unix 系统中，函数调用 getpeername 就可以返回发送端机器的客户端 IP 地址：

```
status = getpeername(tcp_connection_socket,...);
```

但是，使用客户端 IP 地址来识别用户存在着很多缺点，限制了将其作为用户识别技术的效能。

- 客户端 IP 地址描述的是所用的机器，而不是用户。如果多个用户共享同一台计算机，就无法对其进行区分了。
- 很多因特网服务提供商都会在用户登录时为其动态分配 IP 地址。用户每次登录时，都会得到一个不同的地址，因此 Web 服务器不能假设 IP 地址可以在各登录
259 会话之间标识用户。
- 为了提高安全性，并对稀缺的地址资源进行管理，很多用户都是通过网络地址转换（Network Address Translation，NAT）防火墙来浏览网络内容的。这些 NAT 设备隐藏了防火墙后面那些实际客户端的 IP 地址，将实际的客户端 IP 地址转换成了一个共享的防火墙 IP 地址（和不同的端口号）。
- HTTP 代理和网关通常会打开一些新的、到原始服务器的 TCP 连接。Web 服务器看到的将是代理服务器的 IP 地址，而不是客户端的。有些代理为了绕过这个问题会添加特殊的 `Client-IP` 或 `X-Forwarded-For` 扩展首部来保存原始的 IP 地址（参见图 11-1）。但并不是所有的代理都支持这种行为。

有些 Web 站点仍然使用客户端 IP 地址在会话之间跟踪用户的行为，但这种站点并不多。无法用 IP 地址确定目标的地方太多了。

注 1：稍后我们会看到，有些代理确实会添加一个 `Client-IP` 首部，但这并不是 HTTP 标准的一部分。

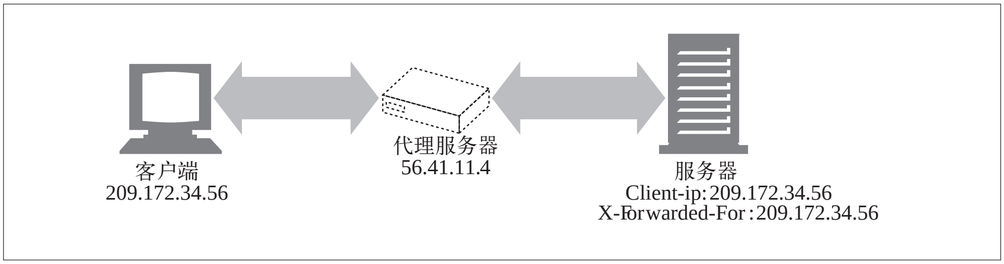


图 11-1　代理可以添加扩展首部，来传递原始客户端的 IP 地址

少数站点甚至将客户端 IP 地址作为一种安全特性使用，它们只向来自特定 IP 地址的用户提供文档。在内部网络中可能可以这么做，但在因特网上就不行了，主要是因为因特网上 IP 地址太容易被欺骗（伪造）了。路径上如果有拦截代理也会破坏此方案。第 14 章讨论了一些强大得多的特权文档访问控制策略。

11.4　用户登录

Web 服务器无需被动地根据用户的 IP 地址来猜测他的身份，它可以要求用户通过用户名和密码进行认证（登录）来显式地询问用户是谁。

为了使 Web 站点的登录更加简便，HTTP 中包含了一种内建机制，可以用 `WWW-Authenticate` 首部和 `Authorization` 首部向 Web 站点传送用户的相关信息。一旦登录，浏览器就可以不断地在每条发往这个站点的请求中发送这个登录信息了，这样，就总是有登录信息可用了。我们将在第 12 章对这种 HTTP 认证机制进行更加详细的讨论，现在我们先来简单地看一看。 260

如果服务器希望在为用户提供对站点的访问之前，先行登录，可以向浏览器回送一条 HTTP 响应代码 401 Login Required。然后，浏览器会显示一个登录对话框，并用 `Authorization` 首部在下一条对服务器的请求中提供这些信息。[2] 图 11-2 对此进行了说明。

此图中发生的情况如下所述。

- 在图 11-2a 中，浏览器对站点 www.joes-hardware.com 发起了一条请求。
- 站点并不知道这个用户的身份，因此在图 11-2b 中，服务器会返回 401 Login Required HTTP 响应码，并添加 `WWW-Authentication` 首部，要求用户登录。这样浏览器就会弹出一个登录对话框。 261

注 2：为了不让用户每发送一条请求都要登录一次，大多数浏览器都会记住某站点的登录信息，并将登录信息放在发送给该站点的每条请求中。

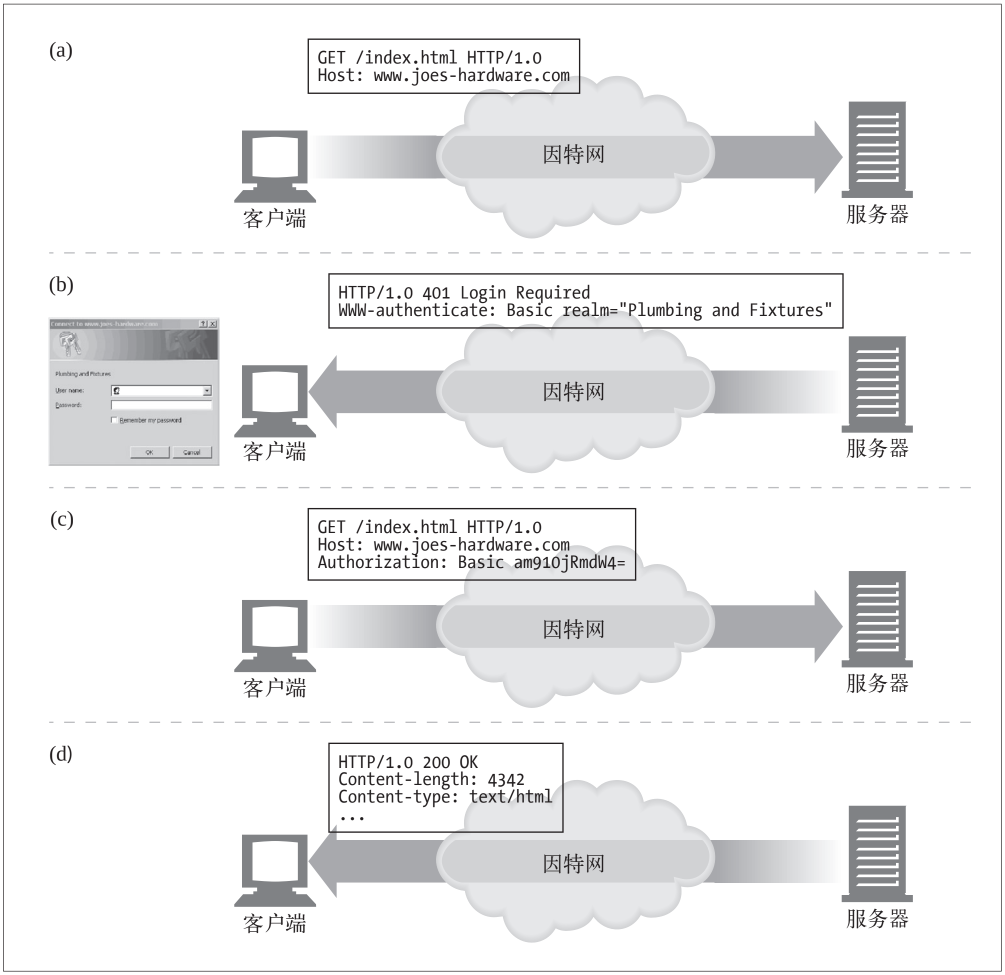


图 11-2　用 HTTP 认证首部注册用户名

- 只要用户输入了用户名和密码（对其身份进行完整性检查），浏览器就会重复原来的请求。这次它会添加一个 `Authorization` 首部，说明用户名和密码。对用户名和密码进行加密，防止那些有意无意的网络观察者看到。[3]
- 现在，服务器已经知道用户的身份了。
- 今后的请求要使用用户名和密码时，浏览器会自动将存储下来的值发送出去，甚至在站点没有要求发送的时候也经常会向其发送。浏览器在每次请求中都向服务器发送 `Authorization` 首部作为一种身份的标识，这样，只要登录一次，就可以在整个会话期间维持用户的身份了。

注 3：在第 14 章我们会看到，任何有这种想法的人，不用费多大事就可以轻易地将 HTTP 基本的认证用户名和密码破解出来。稍后将讨论一些更安全的技术。

但是，登录多个 Web 站点是很繁琐的。Fred 从一个站点浏览到另一个站点的时候，需要在每个站点上登录。更糟的是，可怜的 Fred 很可能要为不同的站点记住不同的用户名和密码。他访问很多站点的时候，他最喜欢的用户名 fred 可能已经被其他人用过了，而且有些站点为用户名和密码的长度和组成设置了不同的规则。Fred 很快就会放弃上网，回去看奥普拉（Oprah）的脱口秀了。下一节我们来讨论这个问题的解决方案。

11.5 胖URL

有些 Web 站点会为每个用户生成特定版本的 URL 来追踪用户的身份。通常，会对真正的 URL 进行扩展，在 URL 路径开始或结束的地方添加一些状态信息。用户浏览站点时，Web 服务器会动态生成一些超链，继续维护 URL 中的状态信息。

改动后包含了用户状态信息的 URL 被称为胖 URL（fat URL）。下面是 Amazon.com 电子商务网站使用的一些胖 URL 实例。每个 URL 后面都附加了一个用户特有的标识码（在这个例子中就是 002-1145265-8016838），这个标识码有助于在用户浏览商店内容时对其进行跟踪。 262

```
...
<a href="/exec/obidos/tg/browse/-/229220/ref=gr_gifts/002-1145265-
   8016838">All Gifts</a><br>
<a href="/exec/obidos/wishlist/ref=gr_pl1_/002-1145265-8016838">Wish
   List</a><br>
...
<a href="http://s1.amazon.com/exec/varzea/tg/armed-forces/-//ref=gr_
af_/002-1145265-8016838">Salute Our Troops</a><br>
<a href="/exec/obidos/tg/browse/-/749188/ref=gr_p4_/002-1145265-8016838"
   >Free Shipping</a><br>
<a href="/exec/obidos/tg/browse/-/468532/ref=gr_returns/002-1145265-8016838"
   >Easy Returns</a>
...
```

可以通过胖 URL 将 Web 服务器上若干个独立的 HTTP 事务捆绑成一个“会话”或“访问”。用户首次访问这个 Web 站点时，会生成一个唯一的 ID，用服务器可以识别的方式将这个 ID 添加到 URL 中去，然后服务器就会将客户端重新导向这个胖 URL。不论什么时候，只要服务器收到了对胖 URL 的请求，就可以去查找与那个用户 ID 相关的所有增量状态（购物车、简介等），然后重写所有的输出超链，使其成为胖 URL，以维护用户的 ID。

可以在用户浏览站点时，用胖 URL 对其进行识别。但这种技术存在几个很严重的问题。

- *丑陋的 URL*
  浏览器中显示的胖 URL 会给新用户带来困扰。

- *无法共享 URL*

  胖 URL 中包含了与特定用户和会话有关的状态信息。如果将这个 URL 发送给其他人，可能就在无意中将你积累的个人信息都共享出去了。

- *破坏缓存*

  为每个 URL 生成用户特有的版本就意味着不再有可供公共访问的 URL 需要缓存了。

- *额外的服务器负荷*

  服务器需要重写 HTML 页面使 URL 变胖。

- *逃逸口*

  用户跳转到其他站点或者请求一个特定的 URL 时，就很容易在无意中“逃离”胖 URL 会话。只有当用户严格地追随预先修改过的链接时，胖 URL 才能工作。如果用户逃离此链接，就会丢失他的进展（可能是一个已经装满了东西的购物车）信息，得重新开始。

- *在会话间是非持久的*

  除非用户收藏了特定的胖 URL，否则用户退出登录时，所有的信息都会丢失。

11.6 cookie

cookie 是当前识别用户，实现持久会话的最好方式。前面各种技术中存在的很多问题对它们都没什么影响，但是通常会将它们与那些技术共用，以实现额外的价值。cookie 最初是由网景公司开发的，但现在所有主要的浏览器都支持它。

cookie 非常重要，而且它们定义了一些新的 HTTP 首部，所以我们要比前面那些技术更详细地介绍它们。cookie 的存在也影响了缓存，大多数缓存和浏览器都不允许
263 对任何 cookie 的内容进行缓存。后面的小节对此进行了更为详细的介绍。

11.6.1 cookie的类型

可以笼统地将 cookie 分为两类：会话 cookie 和持久 cookie。会话 cookie 是一种临时 cookie，它记录了用户访问站点时的设置和偏好。用户退出浏览器时，会话 cookie 就被删除了。持久 cookie 的生存时间更长一些；它们存储在硬盘上，浏览器退出，计算机重启时它们仍然存在。通常会用持久 cookie 维护某个用户会周期性访问的站点的配置文件或登录名。

会话 cookie 和持久 cookie 之间唯一的区别就是它们的过期时间。稍后我们会看到，如果设置了 `Discard` 参数，或者没有设置 `Expires` 或 `Max-Age` 参数来说明扩展的

过期时间，这个 cookie 就是一个会话 cookie。

11.6.2　cookie是如何工作的

cookie 就像服务器给用户贴的“嗨，我叫”的贴纸一样。用户访问一个 Web 站点时，这个 Web 站点就可以读取那个服务器贴在用户身上的所有贴纸。

用户首次访问 Web 站点时，Web 服务器对用户一无所知（参见图 11-3a）。Web 服务器希望这个用户会再次回来，所以想给这个用户“拍上”一个独有的 cookie，这样以后它就可以识别出这个用户了。cookie 中包含了一个由名字 = 值（name=value）这样的信息构成的任意列表，并通过 `Set-Cookie` 或 `Set-Cookie2` HTTP 响应（扩展）首部将其贴到用户身上去。

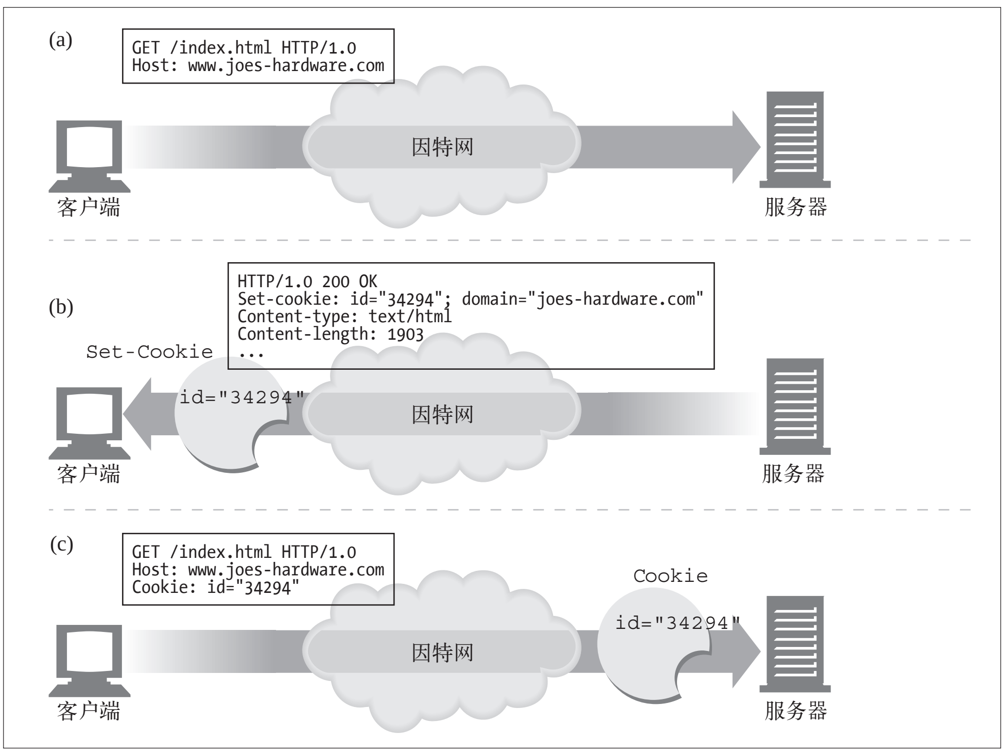


图 11-3　给用户贴一个 cookie

cookie 中可以包含任意信息，但它们通常都只包含一个服务器为了进行跟踪而产生的独特的识别码。比如，在图 11-3b 中，服务器会将一个表示 `id="34294"` 的 cookie 贴到用户上去。服务器可以用这个数字来查找服务器为其访问者积累的数据库信息（购物历史、地址信息等）。

但是，cookie 并不仅限于 ID 号。很多 Web 服务器都会将信息直接保存在 cookie 中。比如：

```
Cookie: name="Brian Totty"; phone="555-1212"
```

浏览器会记住从服务器返回的 `Set-Cookie` 或 `Set-Cookie2` 首部中的 cookie 内容，并将 cookie 集存储在浏览器的 cookie 数据库中（把它当作一个贴有不同国家贴纸的旅行箱）。将来用户返回同一站点时（参见图 11-3c），浏览器会挑中那个服务器贴到用户上的那些 cookie，并在一个 cookie 请求首部中将其传回去。

11.6.3 cookie罐：客户端的状态

cookie 的基本思想就是让浏览器积累一组服务器特有的信息，每次访问服务器时都
264 将这些信息提供给它。因为浏览器要负责存储 cookie 信息，所以此系统被称为客户端侧状态（client-side state）。这个 cookie 规范的正式名称为 HTTP 状态管理机制（HTTP state management mechanism）。

1. 网景的Navigator的cookie

不同的浏览器会以不同的方式来存储 cookie。网景的 Navigator 会将 cookie 存储在
265 一个名为 cookies.txt 的文本文件中。例如：

```
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file! Do not edit.
#
# domain                    allh  path      secure expires     name         value

www.fedex.com               FALSE /         FALSE  1136109676  cc           /us/
.bankofamericaonline.com    TRUE  /         FALSE  1009789256  state        CA
.cnn.com                    TRUE  /         FALSE  1035069235  SelEdition   www
secure.eepulse.net          FALSE /eePulse  FALSE  1007162968  cid          %FE%FF%002
www.reformamt.org           TRUE  /forum    FALSE  1033761379  LastVisit    1003520952
www.reformamt.org           TRUE  /forum    FALSE  1033761379  UserName     Guest
  ...
```

文本文件中的每一行都代表一个 cookie。有 7 个用 tab 键分隔的字段。

- domain（域）
  cookie 的域。
- allh
  是域中所有的主机都获取 cookie，还是只有指定了名字的主机获取。
- path（路径）
  域中与 cookie 相关的路径前缀。

- secure（安全）
  是否只有在使用 SSL 连接时才发送这个 cookie。
- expiration（过期）
  从格林尼治标准时间 1970 年 1 月 1 日 00:00:00 开始的 cookie 过期秒数。
- name（名字）
  cookie 变量的名字。
- value（值）
  cookie 变量的值。

2. 微软Internet Explorer的cookie

微软的 Internet Explorer 将 cookie 存储在高速缓存目录下独立的文本文件中。可以通过浏览这个目录来查看 cookie，如图 11-4 所示。Internet Explorer 中 cookie 文件格式是特有的，但很多字段都很容易理解。cookie 一个接一个地存储在文件中，每个 cookie 都由多行构成。

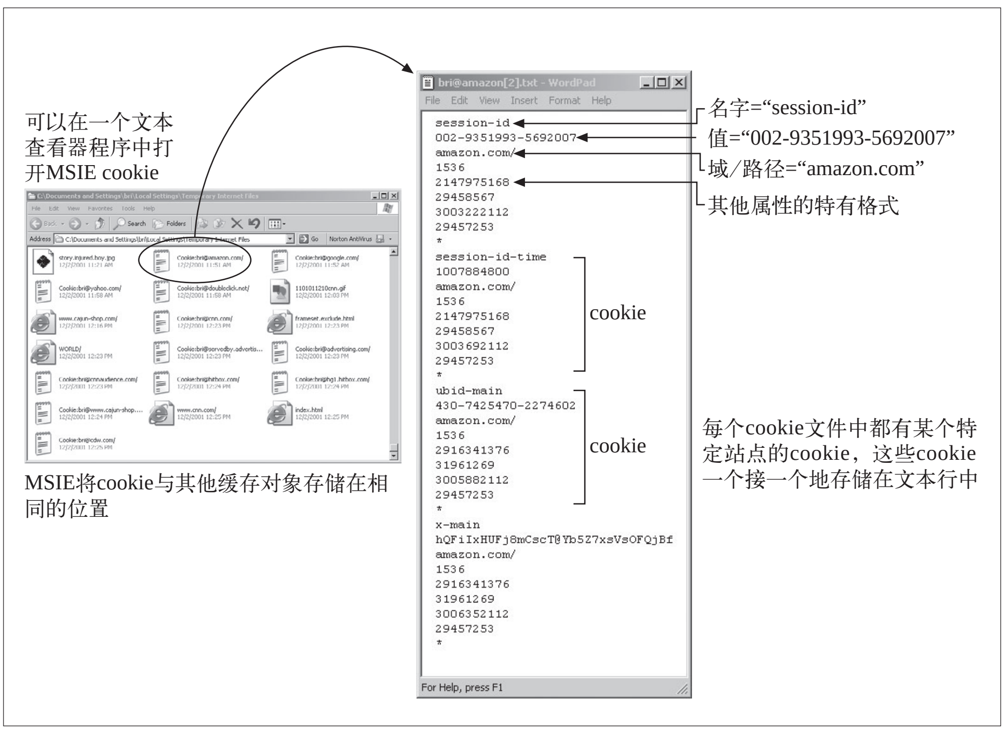


图 11-4　Internet Explorer 的 cookie 存储在缓存目录下独立的文本文件中

文件中每个 cookie 的第一行中都包含了 cookie 的变量名。下一行是变量的值。第三行是域和路径。剩下的行就是一些特有的数据，可能包含日期和一些标记。

11.6.4　不同站点使用不同的cookie

浏览器内部的 cookie 罐中可以有成百上千个 cookie，但浏览器不会将每个 cookie 都发送给所有的站点。实际上，它们通常只向每个站点发送 2 ～ 3 个 cookie。原因如下。

- 对所有这些 cookie 字节进行传输会严重降低性能。浏览器实际传输的 cookie 字节数要比实际的内容字节数多！
- cookie 中包含的是服务器特有的名值对，所以对大部分站点来说，大多数 cookie 都只是无法识别的无用数据。
- 将所有的 cookie 发送给所有站点会引发潜在的隐私问题，那些你并不信任的站
266 点也会获得你只想发给其他站点的信息。

总之，浏览器只向服务器发送服务器产生的那些 cookie。joes-hardware.com 产生的 cookie 会被发送给 joes-hardware.com，不会发送给 bobs-books.com 或 marys-movies.com。

很多 Web 站点都会与第三方厂商达成协议，由其来管理广告。这些广告被做得像 Web 站点的一个组成部分，而且它们确实发送了持久 cookie。用户访问另一个由同一广告公司提供服务的站点时，（由于域是匹配的）浏览器就会再次回送早先设置的持久 cookie。营销公司可以将此技术与 `Referer` 首部结合，暗地里构建一个用户档案和浏览习惯的详尽数据集。现代的浏览器都允许用户对隐私特性进行设置，以限制第三方 cookie 的使用。

1. cookie的域属性

产生 cookie 的服务器可以向 `Set-Cookie` 响应首部添加一个 Domain 属性来控制哪些站点可以看到那个 cookie。比如，下面的 HTTP 响应首部就是在告诉浏览器将
267 `cookie user="mary17"` 发送给域 `".airtravelbargains.com"` 中的所有站点：

```
Set-cookie: user="mary17"; domain="airtravelbargains.com"
```

如果用户访问的是 www.airtravelbargains.com、specials.airtravelbargains.com 或任意以 .airtravelbargains.com 结尾的站点，下列 `Cookie` 首部都会被发布出去：

```
Cookie: user="mary17"
```

2. cookie路径属性

cookie 规范甚至允许用户将 cookie 与部分 Web 站点关联起来。可以通过 Path 属性来实现这一功能，在这个属性列出的 URL 路径前缀下所有 cookie 都是有效的。

例如，某个 Web 服务器可能是由两个组织共享的，每个组织都有独立的 cookie。站点 www.airtravelbargains.com 可能会将部分 Web 站点用于汽车租赁——比如，http://www.airtravelbargains.com/autos/——用一个独立的 cookie 来记录用户喜欢的汽车尺寸。可能会生成一个如下所示的特殊汽车租赁 cookie：

```
Set-cookie: pref=compact; domain="airtravelbargains.com"; path=/autos/
```

如果用户访问 http://www.airtravelbargains.com/specials.html，就只会获得这个 cookie：

```
Cookie: user="mary17"
```

但如果访问 http://www.airtravelbargains.com/autos/cheapo/index.html，就会获得这两个 cookie：

```
Cookie: user="mary17"
Cookie: pref=compact
```

因此，cookie 就是由服务器贴到客户端上，由客户端维护的状态片段，只会回送给那些合适的站点。下面我们来更仔细地看看 cookie 的技术和标准。

11.6.5 cookie成分

现在使用的 cookie 规范有两个不同的版本：cookies 版本 0（有时被称为 Netscape cookies）和 cookies 版本 1（RFC 2965）。cookies 版本 1 是对 cookies 版本 0 的扩展，应用不如后者广泛。

cookie 规范版本 0 和版本 1 都不是作为 HTTP/1.1 规范的一部分提供的。表 11-2 对两个主要的附属文档进行了总结，这两个文档对 cookie 的使用进行了很好的描述。

表11-2 cookie规范

标　题	描　述	位　置
持久客户端状态：HTTP cookies	最初的 Netscape cookie 标准	http://home.netscape.com/newsref/std/cookie_spec.html
RFC 2965：HTTP 状态管理机制	2000 年 10 月的 cookie 标准，废弃了 RFC 2109	http://www.ietf.org/rfc/rfc2965.txt

268

11.6.6　cookies版本0（Netscape）

最初的 cookie 规范是由网景公司定义的。这些“版本 0”的 cookie 定义了 Set-Cookie 响应首部、cookie 请求首部以及用于控制 cookie 的字段。版本 0 的 cookie 看起来如下所示：

```
Set-Cookie: name=value [; expires=date] [; path=path] [; domain=domain]
[; secure]

Cookie: name1=value1 [; name2=value2] ...
```

1. 版本0的Set-Cookie首部

Set-Cookie 首部有一个强制性的 cookie 名和 cookie 值。后面跟着可选的 cookie 属性，中间由分号分隔。表 11-3 描述了 Set-Cookie 字段。

表11-3　版本0（网景）的Set-Cookie属性

Set-Cookie属性	描述及实例
NAME=VALUE	强制的。NAME 和 VALUE 都是字符序列，除非包含在双引号内，否则不包括分号、逗号、等号和空格。Web 服务器可以创建任意的 NAME=VALUE 关联，在后继对站点的访问中会将其送回给 Web 服务器： `Set-Cookie: customer=Mary`

Expires	可选的。这个属性会指定一个日期字符串，用来定义 cookie 的实际生存期。一旦到了过期日期，就不再存储或发布这个 cookie 了。日期的格式为：
	`Weekday, DD-Mon-YY HH:MM:SS GMT`
	唯一合法的时区为 GMT，各日期元素之间的分隔符一定要是长划线。如果没有指定 Expires，cookie 就会在用户会话结束时过期：
	`Set-Cookie: foo=bar; expires=Wednesday, 09-Nov-99 23:12:40 GMT`

Domain	可选的。浏览器只向指定域中的服务器主机名发送 cookie。这样服务器就将 cookie 限制在了特定的域中。acme.com 域就与 anvil.acme.com 和 shipping.crate.acme.com 相匹配，但与 www.cnn.com 就不匹配了。 只有指定域中的主机才能为一个域设置 cookie，这些域中至少要有两个或三个句号，以防止出现 .com、.edu 和 va.us 等形式的域。这里列出了一组固定的特定高层域，落在这个范围中的域只需要两个句号。所有其他域都至少需要三个句号。特定的高层域包括：.com、.edu、.net、.org、.gov、.mil、.int、.biz、.info、.name、.museum、.coop、.aero 和 .pro。 如果没有指定域，就默认为产生 Set-Cookie 响应的服务器的主机名： `Set-Cookie: SHIPPING=FEDEX; domain="joes-hardware.com"`

（续）

Set-Cookie属性	描述及实例
Path	可选的。通过这个属性可以为服务器上特定的文档分配 cookie。如果 Path 属性是一个 URL 路径前缀，就可以附加一个 cookie。路径 /foo 与 /foobar 和 /foo/bar.html 相匹配。路径“/”与域名中所有内容都匹配。 如果没有指定路径，就将其设置为产生 Set-Cookie 响应的 URL 的路径： `Set-Cookie: lastorder=00183; path=/orders`
Secure	可选的。如果包含了这一属性，就只有在 HTTP 使用 SSL 安全连接时才会发送 cookie： `Set-Cookie: private_id=519; secure` 269

2. 版本0的Cookie首部

客户端发送请求时，会将所有与域、路径和安全过滤器相匹配的未过期 cookie 都发送给这个站点。所有 cookie 都被组合到一个 Cookie 首部中：

```
Cookie: session-id=002-1145265-8016838; session-id-time=1007884800
```

11.6.7 cookies版本1（RFC 2965）

RFC 2965（以前的 RFC 2109）定义了一个 cookie 的扩展版本。这个版本 1 标准引入了 Set-Cookie2 首部和 Cookie2 首部，但它也能与版本 0 系统进行互操作。

RFC 2965 cookie 标准比原始的网景公司的标准略微复杂一些，还未得到完全的支持。RFC 2965 cookie 的主要改动包括下列内容。

- 为每个 cookie 关联上解释性文本，对其目的进行解释。
- 允许在浏览器退出时，不考虑过期时间，将 cookie 强制销毁。
- 用相对秒数，而不是绝对日期来表示 cookie 的 Max-Age。
- 通过 URL 端口号，而不仅仅是域和路径来控制 cookie 的能力。
- 通过 Cookie 首部回送域、端口和路径过滤器（如果有的话）。
- 为实现互操作性使用的版本号。
- 在 Cookie 首部从名字中区分出附加关键字的 $ 前缀。

cookie 版本 1 的语法如下所示： 270

```
set-cookie      =     "Set-Cookie2:" cookies
cookies         =     1#cookie
cookie          =     NAME "=" VALUE *(";" set-cookie-av)
NAME            =     attr
VALUE           =     value
set-cookie-av   =     "Comment" "=" value
```

```
                |       "CommentURL" "=" <"> http_URL <">
                |       "Discard"
                |       "Domain" "=" value
                |       "Max-Age" "=" value
                |       "Path" "=" value
                |       "Port" [ "=" <"> portlist <"> ]
                |       "Secure"
                |       "Version" "=" 1*DIGIT
portlist        =       1#portnum
portnum         =       1*DIGIT

cookie          =       "Cookie:" cookie-version 1*((";" | ",") cookie-
                            value)
cookie-value    =      NAME "=" VALUE [";" path] [";" domain] [";" port]
cookie-version  =       "$Version" "=" value
NAME            =       attr
VALUE           =       value
path            =       "$Path" "=" value
domain          =       "$Domain" "=" value
port            =       "$Port" [ "=" <"> value <"> ]
cookie2         =       "Cookie2:" cookie-version
```

1. 版本1的Set-Cookie2首部

版本 1 的 cookie 标准比网景公司标准的可用属性要多。表 11-4 对这些属性做了快速汇总。更详细的解释请参见 RFC 2965。

表11-4　版本1（RFC 2965）的Set-Cookie2属性

Set-Cookie2属性	描述及实例
NAME=VALUE	强制的。Web 服务器可以创建任意的 NAME=VALUE 关联，可以在后继对站点的访问中将其发回给 Web 服务器。“$”是保留字符，所以名字一定不能以它开头
Version	强制的。这个属性的值是一个整数，对应于 cookie 规范的版本。RFC 2965 为版本 1： Set-Cookie2: Part="Rocket_Launcher_0001"; Version="1"
Comment	可选。这个属性说明了服务器准备如何使用这个 cookie。用户可以通过检查此策略来确定是否允许使用带有这个 cookie 的会话。这个值必须采用 UTF-8 编码
CommentURL	可选。这个属性提供了一个 URL 指针，指向详细描述了 cookie 目的及策略的文档。用户可以通过查看此策略来判定是否允许使用带有这个 cookie 的会话
Discard	可选。如果提供了这个属性，就会在客户端程序终止时，指示客户端放弃这个 cookie
Domain	可选。浏览器只向指定域中的服务器主机名发送 cookie。这样服务器就可以将 cookie 限制在特定域中了。acme.com 域与主机名 anvil.acme.com 和 shipping.crate.acme.com 相匹配，但不匹配于 www.cnn.com。域名匹配的规则基本上与 Netscape cookie 一样，但有几条附加的规则。细节请参见 RFC 2965

（续）

Set-Cookie2属性	描述及实例
Max-Age	可选。这个属性的值是一个整数，用于设置以秒为单位的 cookie 生存期。客户端应该根据 HTTP/1.1 的使用期计算规则来计算 cookie 的使用期。cookie 的使用期比 Max-Age 大时，客户端就应该将这个 cookie 丢弃。值为零说明应该立即将那个 cookie 丢弃
Path	可选。通过这个属性可以为服务器上的特定文档指定 cookie。如果 Path 属性是一个 URL 路径的前缀，就可以附加一个 cookie。路径 /foo 匹配于 /foobar 和 /foo/bar.html。路径"/"匹配于域中所有内容。如果没有指定路径，就将其设置为生成 Set-Cookie 响应的 URL 的路径

Port	可选。这个属性可以单独作为关键字使用，也可以包含一个由逗号分隔的、可以应用 cookie 的端口列表。如果有端口列表，就只能向端口与列表中的端口相匹配的服务器提供 cookie。如果单独提供关键字 Port 而没有值，就只能向当前响应服务器的端口号提供 cookie：
	`Set-Cookie2: foo="bar"; Version="1"; Port="80,81,8080"`
	`Set-Cookie2: foo="bar"; Version="1"; Port`

Secure	可选。如果包含这个属性，就只有在 HTTP 使用 SSL 安全连接时才能发送 cookie

271

2. 版本1的Cookie首部

版本 1 的 cookie 会带回与传送的每个 cookie 相关的附加信息，用来描述每个 cookie 途径的过滤器。每个匹配的 cookie 都必须包含来自相应 Set-Cookie2 首部的所有 Domain、Port 或 Path 属性。

比如，假设客户端以前曾收到下列五个来自 Web 站点 www.joes-hardware.com 的 Set-Cookie2 响应：

```
Set-Cookie2: ID="29046"; Domain=".joes-hardware.com"
Set-Cookie2: color=blue
Set-Cookie2: support-pref="L2"; Domain="customer-care.joes-hardware.
com"
Set-Cookie2: Coupon="hammer027"; Version="1"; Path="/tools"
Set-Cookie2: Coupon="handvac103"; Version="1"; Path="/tools/cordless"
```

如果客户端对路径 /tools/cordless/specials.html 又发起了一次请求，就会同时发送这样一个很长的 Cookie 首部：

```
Cookie: $Version="1";
        ID="29046"; $Domain=".joes-hardware.com";
        color="blue";
        Coupon="hammer027"; $Path="/tools";
        Coupon="handvac103"; $Path="/tools/cordless"
```

注意，所有匹配 cookie 都是和它们的 `Set-Cookie2` 过滤器一同传输的，而且保留关键字都是以美元符号（`$`）开头的。

3. 版本1的Cookie2首部和版本协商

`Cookie2` 请求首部负责在能够理解不同 cookie 规范版本的客户端和服务器之间进行互操作性的协商。`Cookie2` 首部告知服务器，用户 Agent 代理理解新形式的 cookie，并提供了所支持的 cookie 标准版本（将其称为 Cookie-Version 更合适一些）：

```
Cookie2: $Version="1"
```

如果服务器理解新形式的 cookie，就能够识别出 `Cookie2` 首部，并在响应首部发送 `Set-Cookie2`（而不是 `Set-Cookie`）。如果客户端从同一个响应中既获得了 `Set-Cookie` 首部，又获得了 `Set-Cookie2` 首部，就会忽略老的 `Set-Cookie` 首部。

如果客户端既支持版本 0 又支持版本 1 的 cookie，但从服务器获得的是版本 0 的 `Set-Cookie` 首部，就应该带着版本 0 的 `Cookie` 首部发送 cookie。但客户端还应该发送 `Cookie2: $Version=`“`1`”来告知服务器它是可以升级的。

11.6.8 cookie与会话跟踪

可以用 cookie 在用户与某个 Web 站点进行多项事务处理时对用户进行跟踪。电子商务 Web 站点用会话 cookie 在用户浏览时记录下用户的购物车信息。我们以流行
272 的购物网站 Amazon.com 为例。在浏览器中输入 http://www.amazon.com 时，就启动了一个事务链，在这些事务中 Web 服务器会通过一系列的重定向、URL 重写以及 cookie 设置来附加标识信息。

图 11-5 显示了从一次 Amazon.com 访问中捕获的事务序列。

- 图 11-5a——浏览器首次请求 Amazon.com 根页面。
- 图 11-5b——服务器将客户端重定向到一个电子商务软件的 URL 上。
- 图 11-5c——客户端对重定向的 URL 发起一个请求。
- 图 11-5d——服务器在响应上贴上两个会话 cookie，并将用户重定向到另一个 URL，这样客户端就会用这些附加的 cookie 再次发出请求。这个新的 URL 是个胖 URL，也就是说有些状态嵌入到 URL 中去了。如果客户端禁止了 cookie，只要用户一直跟随着 Amazon.com 产生的胖 URL 链接，不离开网站，仍然可以实现一些基本的标识功能。
- 图 11-5e——客户端请求新的 URL，但现在会传送两个附加的 cookie。

- 图 11-5f——服务器重定向到 home.html 页面，并附加另外两个 cookie。
- 图 11-5g——客户端获取 home.html 页面并将所有四个 cookie 都发送出去。
- 图 11-5h——服务器回送内容。

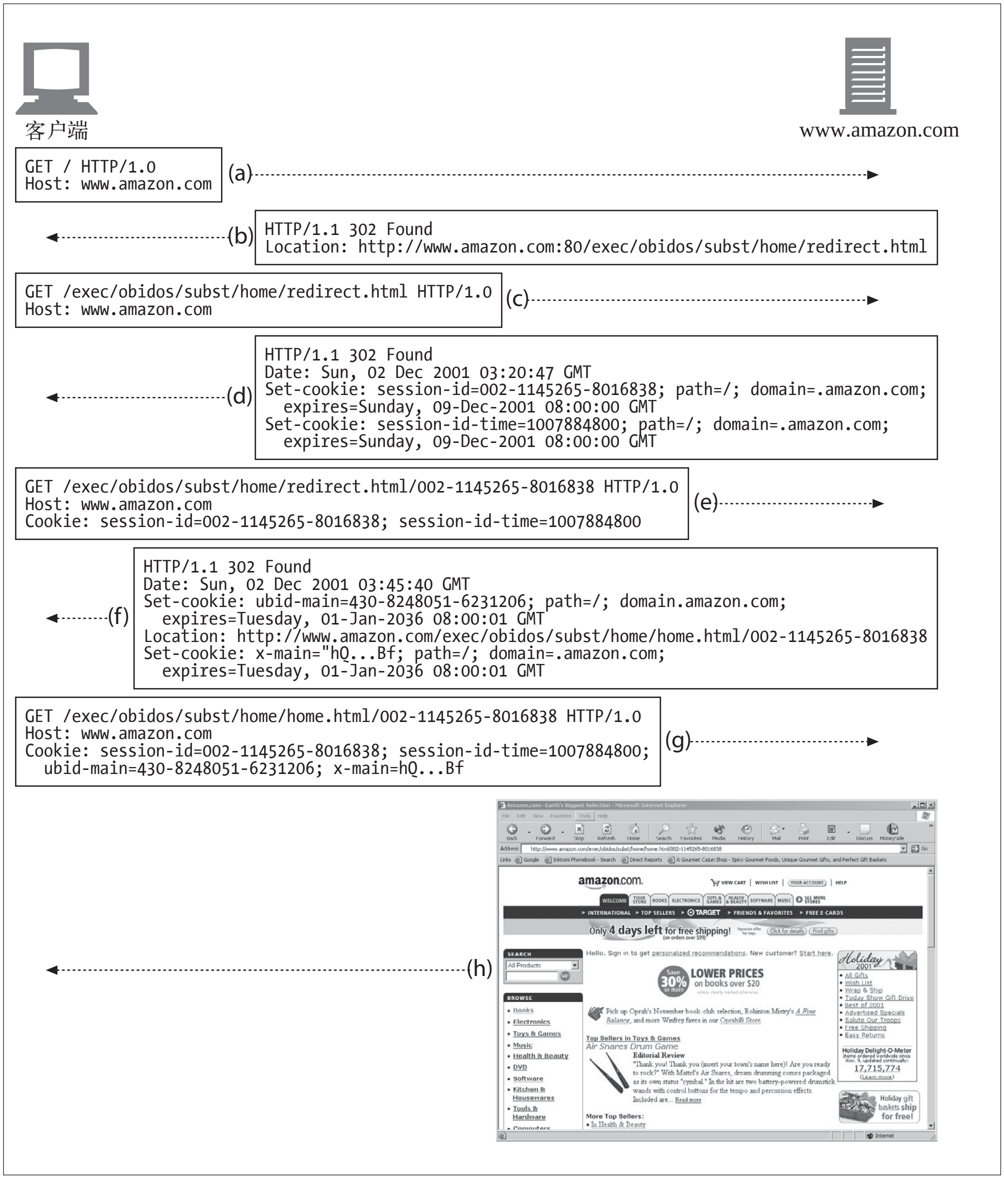


图 11-5　Amazon.com 网站用会话 cookie 来跟踪用户

11.6.9 cookie与缓存

缓存那些与 cookie 事务有关的文档时要特别小心。你不会希望给用户分配一个过去某些用户用过的 cookie，或者更糟糕的是，向一个用户展示其他人私有文档的内容。

cookie 和缓存的规则并没有很好地建立起来。下面是处理缓存时的一些指导性规则。

- 如果无法缓存文档，要将其标示出来

  文档的所有者最清楚文档是否是不可缓存的。如果文档不可缓存，就显式地注明——具体来说，如果除了 `Set-Cookie` 首部之外文档是可缓存的，就使用 `Cache-Control:no-cache="Set-Cookie"`。另一种更通用的做法是为可缓存文档使用 `Cache-Control:public`，这样有助于节省 Web 中的带宽。

- 缓存 `Set-Cookie` 首部时要小心

  如果响应中有 `Set-Cookie` 首部，就可以对主体进行缓存（除非被告知不要这么做），但要特别注意对 `Set-Cookie` 首部的缓存。如果向多个用户发送了相同的 `Set-Cookie` 首部，可能会破坏用户的定位。

  有些缓存在将响应缓存起来之前会删除 `Set-Cookie` 首部，但这样也会引发一
273~274 些问题，因为在没有缓存的时候，通常都会有 cookie 贴在客户端上，但由缓存提供服务的客户端就不会有 cookie 了。强制缓存与原始服务器重新验证每条请求，并将返回的所有 `Set-Cookie` 首部都合并到客户端的响应中去，就可以改善这种状况。原始服务器可以通过向缓存的副本中添加这个首部来要求进行这种再验证：

  ```
  Cache-Control: must-revalidate, max-age=0
  ```

  即便内容实际上是可以缓存的，比较保守的缓存可能也会拒绝缓存所有包含 `Set-Cookie` 首部的响应。有些缓存允许使用缓存 `Set-Cookie` 图片，但不缓存文本的模式。

- 小心处理带有 `Cookie` 首部的请求

  带有 `Cookie` 首部的请求到达时，就在提示我们，得到的结果可能是私有的。一定要将私有内容标识为不可缓存的，但有些服务器可能会犯错，没有将此内容标记为不可缓存的。

  有些响应文档对应于携带 `Cookie` 首部的请求，保守的缓存可能会选择不去缓存这些响应文档。同样，有些缓存允许使用缓存 cookie 图片，而不缓存文本的模式。得到更广泛接受的策略是缓存带有 `Cookie` 首部的图片，将过期时间设置为零，强制每次都进行再验证。

11.6.10　cookie、安全性和隐私

cookie 是可以禁止的，而且可以通过日志分析或其他方式来实现大部分跟踪记录，所以 cookie 自身并不是很大的安全隐患。实际上，可以通过提供一个标准的审查方法在远程数据库中保存个人信息，并将匿名 cookie 作为键值，来降低客户端到服务器的敏感数据传送频率。

但是，潜在的滥用情况总是存在的，所以，在处理隐私和用户跟踪信息时，最好还是要小心一些。第三方 Web 站点使用持久 cookie 来跟踪用户就是一种最大的滥用。将这种做法与 IP 地址和 `Referer` 首部信息结合在一起，这些营销公司就可以构建起相当精确的用户档案和浏览模式信息。

尽管有这么多负面的宣传，人们通常还是认为，如果能够小心地确认在向谁提供私人信息，并仔细查阅站点的隐私政策，那么，cookie 会话处理和事务处理所带来的便利性要比大部分风险更重要。

1998 年，计算机事故咨询能力组织（Computer Incident Advisory Capability）（美国能源部的一部分）编写了一份过分使用 cookie 的风险评估报告。下面是那份报告的摘要。

> CIAC I-034：因特网cookie
>
> （http://www.ciac.org/ciac/bulletins/i-034.shtml）
>
> - 问题
>
> cookie是Web服务器用来识别Web用户的小块数据。关于cookie功能的流行说法和谣言之间的比例已经达到了令人不解的地步，使用户恐惧，使管理者忧心。 275
>
> - 脆弱性评估
>
> 由于使用Web浏览器cookie使得系统被破坏或窃听，从而带来的系统脆弱性本质上并不存在。cookie只能告知Web服务器你以前是否到过某个网站，并在下次访问时将来自Web服务器的一些短小信息（比如用户编码）回送给它。大部分cookie只会持续到用户退出浏览器为止，然后就会被破坏掉。第二种名为持久cookie的cookie有一个过期日期，会在你的硬盘上存储到那个日期为止。无论用户何时返回一个站点，都可以通过持久cookie来识别其身份，以便跟踪用户的浏览习惯。你来自何处，以及访问过哪些Web页面等信息已经存储在Web服务器的日志文件中了，也可以用这些信息来跟踪用户的浏览习惯，只是使用cookie更简单一些罢了。

11.7 更多信息

这里还有几份有用的资源，介绍了更多与 cookie 有关的信息。

- *Cookies*
  Simon St.Laurent 著，McGraw-Hill 公司出版。

- http://www.ietf.org/rfc/rfc2965.txt
  RFC 2965，“HTTP State Management Mechanism”（“HTTP 状态管理机制”）（废弃了 RFC 2109）。

- http://www.ietf.org/rfc/rfc2964.txt
  RFC 2964，“Use of HTTP State Management”（“HTTP 状态管理的用途”）。

- http://home.netscape.com/newsref/std/cookie_spec.html
  这个经典的网景公司文档“Persistent Client State: HTTP Cookies”（“持久的客户端
  276 状态：HTTP Cookies”）描述了现在仍在广泛使用的 HTTP cookie 的最初形式。

第12章

基本认证机制

有数百万的人在用 Web 进行私人事务处理，访问私有的数据。通过 Web 可以很方便地访问这些信息，但仅仅是方便访问还是不够的。我们要保证只有特定的人能看到我们的敏感信息并且能够执行我们的特权事务。并不是所有的信息都能够公开发布的。

未授权用户无法查看我们的在线旅游档案，也不能在未经许可的情况下向 Web 站点发布文档，这会让我们感觉舒服一些。我们还要确保，组织中未经授权或不怀好意的成员无法获取那些最敏感的公司计划文档。我们与孩子、配偶以及暗恋对象的私人 Web 通信都是在带有些许隐私保护的情况下进行的，这样我们才能放心。

服务器需要通过某种方式来了解用户身份。一旦服务器知道了用户身份，就可以判定用户可以访问的事务和资源了。认证就意味着要证明你是谁。通常是通过提供用户名和密码来进行认证的。HTTP 为认证提供了一种原生工具。尽管我们可以在 HTTP 的认证形式和 cookie 基础之上“运行自己的”认证工具，但在很多情况下，HTTP 的原生认证功能就可以很好地满足要求。

本章阐述了 HTTP 的认证机制，深入介绍了最常见的 HTTP 认证形式，*基本认证*（basic authentication）。下一章将介绍一种称为*摘要认证*（digest authentication）的功能更强的认证技术。

12.1　认证

认证就是要给出一些身份证明。当出示像护照或驾照那样有照片的身份证件时，就给出了一些证据，说明你就是你所声称的那个人。在自动取款机上输入 PIN 码，或
277 在计算机系统的对话框中输入了密码时，也是在证明你就是你所声称的那个人。

现在，这些策略都不是绝对有效的。密码可以被猜出来或被人偶然听到，身份证件可能被偷去或被伪造，但每种证据都有助于构建合理的信任，说明你就是你所声称的那个人。

12.1.1　HTTP的质询/响应认证框架

HTTP 提供了一个原生的*质询 / 响应*（challenge/response）框架，简化了对用户的认证过程。HTTP 的认证模型如图 12-1 中所示。

Web 应用程序收到一条 HTTP 请求报文时，服务器没有按照请求执行动作，而是以一个“认证质询”进行响应，要求用户提供一些保密信息来说明他是谁，从而对其进行质询。

用户再次发起请求时，要附上保密证书（用户名和密码）。如果证书不匹配，服务器可以再次质询客户端，或产生一条错误信息。如果证书匹配，就可以正常完成请求了。

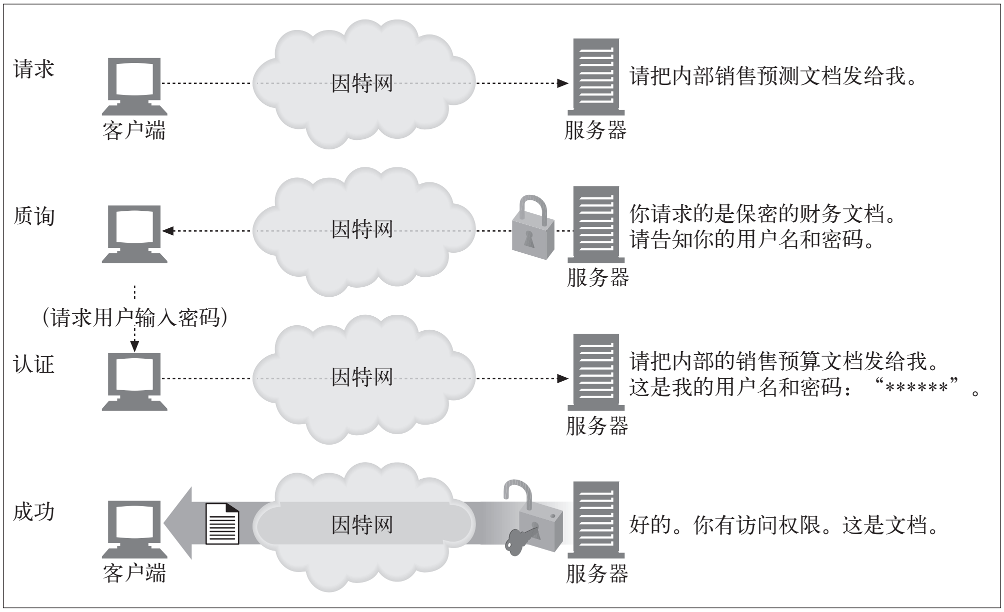


图 12-1　简化的质询 / 响应认证框架

12.1.2　认证协议与首部

HTTP 通过一组可定制的控制首部，为不同的认证协议提供了一个可扩展框架。表 12-1 列出的首部格式和内容会随认证协议的不同而发生变化。认证协议也是在 HTTP 认证首部中指定的。 278

HTTP 定义了两个官方的认证协议：基本认证和摘要认证。今后人们可以随意设计一些使用 HTTP 质询 / 响应框架的新协议。本章的其余部分将解释基本认证机制。摘要认证的细节请参见第 13 章。

表12-1　认证的4个步骤

步骤	首　　部	描　　述	方法/状态
请求		第一条请求没有认证信息	GET
质询	`WWW-Authenticate`	服务器用 401 状态拒绝了请求，说明需要用户提供用户名和密码。 服务器上可能会分为不同的区域，每个区域都有自己的密码，所以服务器会在 `WWW-Authenticate` 首部对保护区域进行描述。 同样，认证算法也是在 `WWW-Authenticate` 首部中指定的	401 Unauthorized

（续）

步骤	首　部	描　述	方法/状态
授权	Authorization	客户端重新发出请求，但这一次会附加一个 Authorization 首部，用来说明认证算法、用户名和密码	GET
成功	Authentication-Info	如果授权证书是正确的，服务器就会将文档返回。有些授权算法会在可选的 Authentication-Info 首部返回一些与授权会话相关的附加信息	200 OK

为了具体地说明这个问题，我们来看看图 12-2。

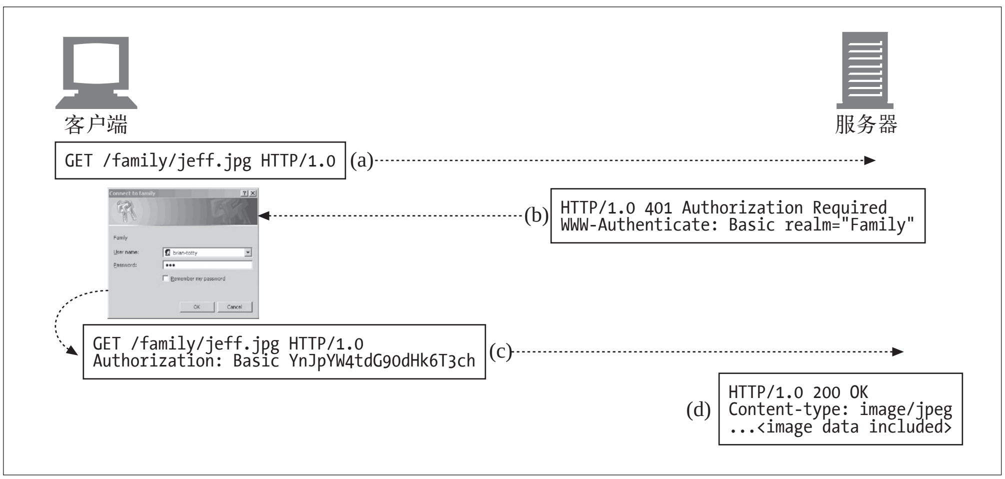


图 12-2　基本认证实例

服务器对用户进行质询时，会返回一条 401 Unauthorized 响应，并在 WWW-Authenticate
279 首部说明如何以及在哪里进行认证（参见图 12-2b）。

当客户端授权服务器继续处理时，会重新发送请求，但会在 Authorization 首部附上加密的密码和其他一些认证参数（参见图 12-2c）。

授权请求成功完成时，服务器会返回一个正常的状态码（比如，200 OK）；对高级认证算法来说，可能还会在 Authentication-Info 首部附加一些额外的信息（参见图 12-2d）。

12.1.3　安全域

在对基本认证的细节进行讨论之前，需要解释一下 HTTP 是怎样允许服务器为

不同的资源使用不同的访问权限的。你可能已经注意到了，图 12-2b 的 `WWW-Authenticate` 质询中包含了一个 realm 指令。Web 服务器会将受保护的文档组织成一个安全域（security realm）。每个安全域都可以有不同的授权用户集。

比如，假设 Web 服务器建立了两个安全域：一个用于公司的财务信息，另一个用于个人家庭文档（参见图 12-3）。不同的用户对各个安全域的访问权限是不同的。公司的 CEO 应该能够访问销售额预测资料，但不应该允许他访问员工和其家人度假的照片！

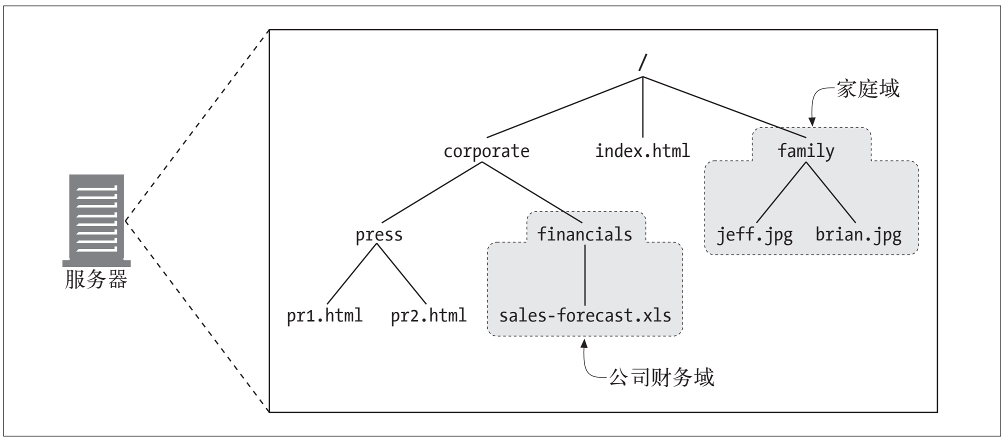


图 12-3　Web 服务器上的安全域

下面是一个假想的基本认证质询，它指定了一个域：

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Basic realm="Corporate Financials"
```

域应该有一个描述性的字符名，比如 Corporate Financials（公司财务资料），以帮助用户了解应该使用哪个用户名和密码。在安全域的名称中列出服务器主机名也是很有帮助的——比如，executive-committee@bigcompany.com。 280

12.2　基本认证

基本认证是最流行的 HTTP 认证协议。几乎每个主要的客户端和服务器都实现了基本认证机制。基本认证最初是在 HTTP/1.0 规范中提出的，但此后被移到了 RFC 2617 中，它详细介绍了 HTTP 的认证机制。

在基本认证中，Web 服务器可以拒绝一个事务，质询客户端，请用户提供有效的用户名和密码。服务器会返回 401 状态码，而不是 200 状态码来初始化认证质询，并

用 WWW-Authenticate 响应首部指定要访问的安全域。浏览器收到质询时，会打开一个对话框，请求用户输入这个域的用户名和密码。然后将用户名和密码稍加扰码，再用 Authorization 请求首部回送给服务器。

12.2.1　基本认证实例

图 12-2 是一个详细的基本认证实例。

- 在图 12-2a 中，用户请求了私人家庭相片 /family/jeff.jpg。
- 在图 12-2b 中，服务器回送一条 401 Authorization Required，对私人家庭相片进行密码质询，同时回送的还有 WWW-Authenticate 首部。这个首部请求对 Family 域进行基本认证。
- 在图 12-2c 中，浏览器收到了 401 质询，弹出对话框，询问 Family 域的用户名和密码。用户输入用户名和密码时，浏览器会用一个冒号将其连接起来，编码成"经过扰码的"Base-64 表示形式（下节介绍），然后将其放在 Authorization 首部中回送。
- 在图 12-2d 中，服务器对用户名和密码进行解码，验证它们的正确性，然后用一条 HTTP 200 OK 报文返回所请求的报文。

表 12-2 总结了 HTTP 基本认证的 WWW-Authenticate 和 Authorization 首部。

表12-2　基本认证首部

质询/响应	首部语法及描述
质询（服务器发往客户端）	网站的不同部分可能有不同的密码。域就是一个引用字符串，用来命名所请求的文档集，这样用户就知道该使用哪个密码了： WWW-Authenticate: Basic realm=quoted-realm
响应（客户端发往服务器）	用冒号（：）将用户名和密码连接起来，然后转换成 Base-64 编码，这样在用户名和密码中包含国际字符会稍微容易一些，也能尽量避免通过观察网络流量并只进行一些粗略的检查就可以获取用户名和密码情况的发生： 281 Authorization: Basic base64-username-and-password

注意，基本认证协议并没有使用表 12-1 所示的 Authentication-Info 首部。

12.2.2　Base-64 用户名/密码编码

HTTP 基本认证将（由冒号分隔的）用户名和密码打包在一起，并用 Base-64 编码方式对其进行编码。如果不知道 Base-64 编码是什么意思，也不用担心。你并不需要对它有太多的了解，如果对此感兴趣，可以在附录 E 中读到所有与之有关的内容。简单来说，Base-64 编码会将一个 8 位字节序列划分为一些 6 位的块。用每个 6

位的块在一个特殊的由 64 个字符组成的字母表中选择一个字符，这个字母表中包含了大部分字母和数字。

图 12-4 显示了使用 Base-64 编码的基本认证实例。在这个例子中，用户名为 brian-totty，密码为 Ow!。浏览器用冒号将用户名和密码连接起来，生成一个打包字符串 brian-totty:Ow!。然后对这个字符串进行 Base-64 编码，变成一串乱码：YnJpYW4tdG90dHk6T3ch。

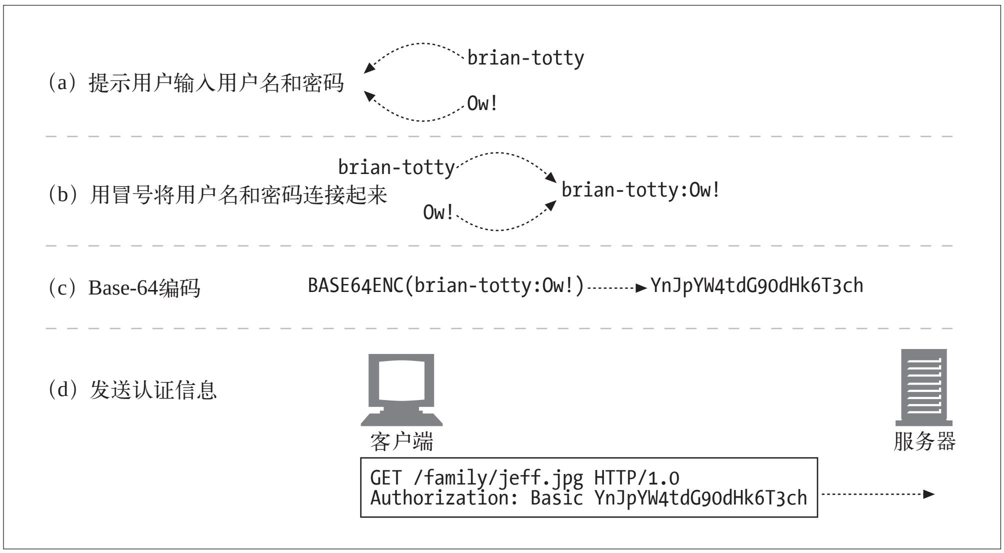


图 12-4　从用户名和密码中生成一个基本认证首部

Base-64 编码可以接受二进制字符串、文本、国际字符表示的数据（在某些系统中会引发一些问题），将其暂时转换成一个易移植的字母表以便传输。然后，在远端就可以解码出原始字符串，而无需担心传输错误了。

有些用户名和密码中会包含国际字符或其他在 HTTP 首部中非法的字符（比如引号、冒号和回车换行符），对这些用户名和密码来说，Base-64 编码是非常有用的。而且，Base-64 编码扰乱了用户名和密码，这样也可以防止管理员在管理服务器和网络时，不小心看到用户名和密码。 282

12.2.3　代理认证

中间的代理服务器也可以实现认证功能。有些组织会在用户访问服务器、LAN 或无线网络之前，用代理服务器对其进行认证。可以在代理服务器上对访问策略进行集中管理，因此，通过代理服务器提供对某组织内部资源的统一访问控制是一种很便捷的

方式。这个过程的第一步就是通过代理认证（proxy authentication）来识别身份。

代理认证的步骤与 Web 服务器身份验证的步骤相同。但首部和状态码都有所不同。表 12-3 列出了 Web 服务器和代理在认证中使用的状态码和首部的差异。

表12-3　Web服务器与代理认证

Web服务器	代理服务器
Unauthorized status code: 401	Unauthorized status code: 407
WWW-Authenticate	Proxy-Authenticate
Authorization	Proxy-Authorization
Authentication-Info	Proxy-Authentication-Info

12.3　基本认证的安全缺陷

基本认证简单便捷，但并不安全。只能用它来防止非恶意用户无意间进行的访问，或将其与 SSL 这样的加密技术配合使用。

基本认证存在下列安全缺陷。

(1) 基本认证会通过网络发送用户名和密码，这些用户名和密码都是以一种很容易解码的形式表示的。实际上，密码是以明文形式传输的，任何人都可以读取并将其捕获。虽然 Base-64 编码通过隐藏用户名和密码，致使友好的用户不太可能在进行网络观测时无意中看到密码，但 Base-64 编码的用户名和密码可以很轻易地通过反向编码过程进行解码，甚至可以用纸笔在几秒钟内手工对其进行解码！所以经过 Base-64 编码的密码实际上就是“明文”传送的。如果有动机的第三方用户有可能会去拦截基本认证发送的用户名和密码，就要通过 SSL 加密信道发送
283 所有的 HTTP 事务，或者使用更安全的认证协议，比如摘要认证。

(2) 即使密码是以更难解码的方式加密的，第三方用户仍然可以捕获被修改过的用户名和密码，并将修改过的用户名和密码一次一次地重放给原始服务器，以获得对服务器的访问权。没有什么措施可用来防止这些重放攻击。

(3) 即使将基本认证用于一些不太重要的应用程序，比如公司内部网络的访问控制或个性化内容的访问，一些不良习惯也会让它变得很危险。很多用户由于受不了大量密码保护的服务，会在这些服务间使用相同的用户名和密码。比如说，某个狡猾的恶徒会从免费的因特网邮件网站捕获明文形式的用户名和密码，然后会发现用同样的用户名和密码还可以访问重要的在线银行网站！

(4) 基本认证没有提供任何针对代理和作为中间人的中间节点的防护措施，它们没有修改认证首部，但却修改了报文的其余部分，这样就严重地改变了事务的本质。

(5) 假冒服务器很容易骗过基本认证。如果在用户实际连接到一台恶意服务器或网关的时候，能够让用户相信他连接的是一个受基本认证保护的合法主机，攻击者就可以请求用户输入密码，将其存储起来以备未来使用，然后捏造一条错误信息传送给用户。

这一切说明，在友好的环境，或者说是希望有隐私保护但隐私保护并不十分必要的环境中，可以通过基本认证来提供便捷的文档个性化服务或访问控制保护。通过这种方式，可以用基本认证来防止一些好奇的用户无意中或不小心对文档进行访问。[1]

比如，在一个公司内部，产品管理可能要对未来的产品计划进行密码保护，以防止信息的过早发布。对一般用户而言，基本认证就足以让他们感到不便而不会再去访问这些数据了。[2] 同样，你可能会用密码来保护那些并非高度机密的，或者没什么信息价值的私人照片或私有站点，这些信息确实和其他人也没什么关系。

将基本认证与加密数据传输（比如 SSL）配合使用，向恶意用户隐藏用户名和密码，会使基本认证变得更加安全。这是一种常用的技巧。

我们会在第 14 章讨论安全加密技术。下一章将介绍更复杂的 HTTP 认证协议——摘要认证，摘要认证具有比基本认证更强的安全特性。 284

12.4 更多信息

更多与基本认证和 LDAP 有关的信息，请参见以下资源。

- http://www.ietf.org/rfc/rfc2617.txt
  RFC 2617，“HTTP Authentication：Basic and Digest Access Authentication.”（“HTTP 认证：基本和摘要访问认证”）

- http://www.ietf.org/rfc/rfc2616.txt 285
  RFC 2616，“Hypertext Transfer Protocol-HTTP/1.1.”（“超文本传输协议——HTTP/1.1”。）

注 1：小心，基本认证中使用的用户名和密码要有别于你在更安全的系统中所使用的密码，否则恶意用户就可以用它们来攻破你的安全账户了！

注 2：尽管不是非常安全，但公司内部的员工通常也没有太大的动力去恶意捕获这些密码。这也说明，公司确实会有间谍，也确实会有不满，想要报复的员工，所以，明智的做法是对一旦被恶意获取就会造成很大损害的数据应用更安全的策略。

第13章

摘要认证

基本认证便捷灵活，但极不安全。用户名和密码都是以明文形式传送的，[1] 也没有采取任何措施防止对报文的篡改。安全使用基本认证的唯一方式就是将其与 SSL 配合使用。

摘要认证与基本认证兼容，但却更为安全。本章主要介绍摘要认证的原理和实际应用。尽管摘要认证还没有得到广泛应用，但对实现安全事务来说，这些概念是非常重要的。

13.1 摘要认证的改进

摘要认证是另一种 HTTP 认证协议，它试图修复基本认证协议的严重缺陷。具体来说，摘要认证进行了如下改进。

- 永远不会以明文方式在网络上发送密码。
- 可以防止恶意用户捕获并重放认证的握手过程。
- 可以有选择地防止对报文内容的篡改。
- 防范其他几种常见的攻击方式。

摘要认证并不是最安全的协议。[2] 摘要认证并不能满足安全 HTTP 事务的很多需求。对这些需求来说，使用传输层安全（Transport Layer Security，TLS）和安全 HTTP
286 （Secure HTTP，HTTPS）协议更为合适一些。

但摘要认证比它要取代的基本认证强大很多。与很多建议其他因特网服务使用的常用策略相比，（比如曾建议 LDAP、POP 和 IMAP 使用的 CRAM-MD5），摘要认证也要强大很多。

迄今为止，摘要认证还没有被广泛应用。但由于基本认证存在固有的安全风险，HTTP 设计者曾在 RFC 2617 中建议："在可行的情况下应该将目前在用的所有使用基本认证的服务，尽快地转换为摘要认证方式。"[3] 这个标准的前景还不太明朗。

13.1.1 用摘要保护密码

摘要认证遵循的箴言是"绝不通过网络发送密码"。客户端不会发送密码，而是会发送一个"指纹"或密码的"摘要"，这是密码的不可逆扰码。客户端和服务器都知道这个密码，因此服务器可以验证所提供的摘要是否与密码相匹配。只拿到摘要的话，除了将所有的密码都拿来试试之外，没有其他方法可以找出摘要是来自哪个密

注 1：用户名和密码用 Base-64 编码进行了扰码，但很容易被解码。只能防止无意中的查看，没有任何防止恶意用户攻击的手段。

注 2：比如，与基于公有密钥的机制相比，摘要认证所提供的认证机制就不够强。同样，摘要认证除了能保护密码外，并没有提供保护其他内容的方式——请求和应答中的其余部分仍然可能被窃听。

注 3：随着 SSL 加密 HTTP 的流行和广泛采用，有关摘要认证的现实意义曾有过很激烈的争论。时间将会告诉我们摘要认证能否达到所需的规模。

码的！[4]

下面来看看摘要认证的工作原理（这是一个简化版本）。

- 在图 13-1a 中，客户端请求了某个受保护文档。
- 在图 13-1b 中，在客户端能够证明其知道密码从而确认其身份之前，服务器拒绝提供文档。服务器向客户端发起质询，询问用户名和摘要形式的密码。
- 在图 13-1c 中，客户端传递了密码的摘要，证明它是知道密码的。服务器知道所有用户的密码，[5] 因此可以将客户提供的摘要与服务器自己计算得到的摘要进行比较，以验证用户是否知道密码。另一方在不知道密码的情况下，很难伪造出正确的摘要。
- 在图 13-1d 中，服务器将客户端提供的摘要与服务器内部计算出的摘要进行对比。如果匹配，就说明客户端知道密码（或者很幸运地猜中了！）。可以设置摘要函数，使其产生很多数字，让人不可能幸运地猜中摘要。服务器进行了匹配验证之后，会将文档提供给客户端——整个过程都没有在网络上发送密码。 287

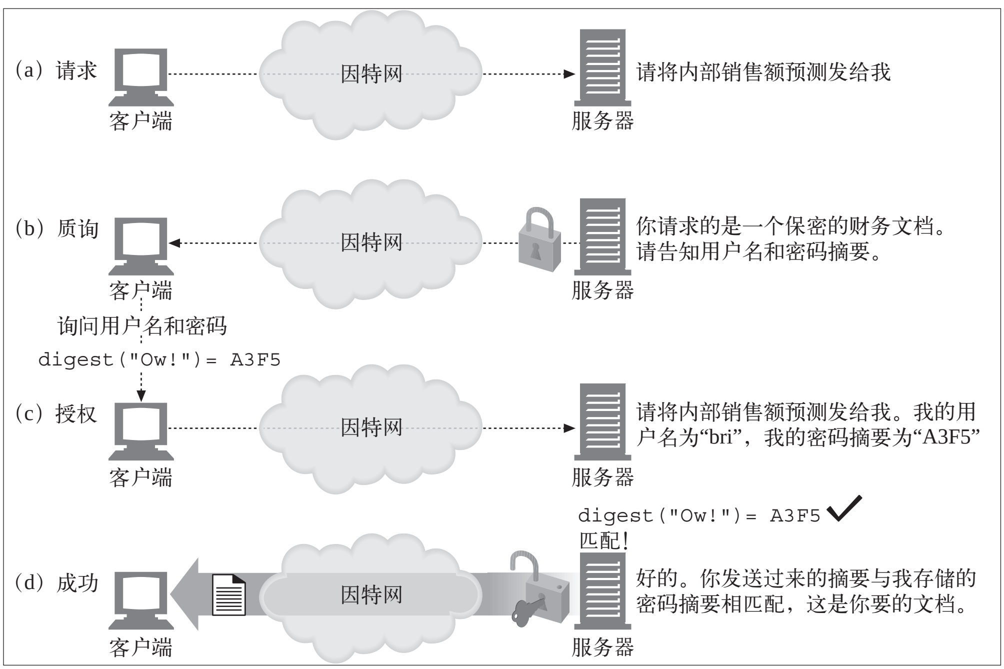


图 13-1　用摘要来实现隐藏密码的认证

注 4：有一些技术，比如词典攻击，会首先尝试一些常见的密码。这些密码分析技术可以极大地简化密码破译进程。

注 5：实际上，服务器只需要知道密码的摘要即可。

我们将在表 13-8 中更详细地讨论摘要认证中那些特殊的首部。

13.1.2 单向摘要

摘要是“对信息主体的浓缩”。[6] 摘要是一种单向函数，主要用于将无限的输入值转换为有限的浓缩输出值。[7] 常见的摘要函数 MD5，[8] 会将任意长度的字节序列转换为一个 128 位的摘要。

128 位 = 2^{128}，或者大约

288 1 000 000 000 000 000 000 000 000 000 000 000 000 000 种不同的输出值。

对这些摘要来说，最重要的是如果不知道密码的话，要想正确地猜出发送给服务器的摘要将是非常困难的。同样，如果有摘要，想要判断出它是由无数输入值中的哪一个产生的，也是非常困难的。

MD5 输出的 128 位的摘要通常会被写成 32 个十六进制的字符，每个字符表示 4 位。表 13-1 给出了几个示例输入的 MD5 摘要。注意 MD5 是怎样根据任意的输入产生定长的摘要输出的。

表13-1 MD5摘要实例

输　　入	MD5摘要
"Hi"	C1A5298F939E87E8F962A5EDFC206918
"bri:Ow!"	BEAAA0E34EBDB072F8627C038AB211F8
"3.1415926535897"	475B977E19ECEE70835BC6DF46F4F6DE
"http://www.http-guide.com/index.htm"	C617C0C7D1D05F66F595E22A4B0EAAA5
"WE hold these Truths to be self-evident, that all Men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the Pursuit of Happiness—That to secure these Rights, Governments are instituted among Men, deriving their just Powers from the Consent of the Governed, that whenever any Form of Government becomes destructive of these Ends, it is the Right of the People to alter or to abolish it, and to institute new Government, laying its Foundation on such Principles, and organizing its Powers in such Form, as to them shall seem most likely to effect their Safety and Happiness."	66C4EF58DA7CB956BD04233FBB64E0A4

注 6：韦氏词典，1998 年。

注 7：理论上来讲，我们将数量无限的输入值转换成了数量有限的输出值，所以两个不同的输入值就可能映射为同一个摘要。这种情况被称为*冲突*（collision）。实际上，由于可用输出值的数量足够大，所以在现实生活中，出现冲突的可能是微乎其微的，对我们要实现的密码匹配来说并不重要。

注 8：MD5 表示“报文摘要的第五版”，是摘要算法系列中的一种。*安全散列算法*（Secure Hash Algorithm，SHA）是另一种常见的摘要函数。

有时也将摘要函数称为加密的校验和、单向散列函数或指纹函数。

13.1.3 用随机数防止重放攻击

使用单向摘要就无需以明文形式发送密码了。可以只发送密码的摘要，而且可以确信，没有哪个恶意用户能轻易地从摘要中解码出原始密码。

但是，仅仅隐藏密码并不能避免危险，因为即便不知道密码，别有用心的人也可以截获摘要，并一遍遍地重放给服务器。摘要和密码一样好用。

为防止此类重放攻击的发生，服务器可以向客户端发送一个称为随机数（nonce）[9]的特殊令牌，这个数会经常发生变化（可能是每毫秒，或者是每次认证都变化）。客户端在计算摘要之前要先将这个随机数令牌附加到密码上去。 289

在密码中加入随机数就会使摘要随着随机数的每一次变化而变化。记录下的密码摘要只对特定的随机值有效，而没有密码的话，攻击者就无法计算出正确的摘要，这样就可以防止重放攻击的发生。

摘要认证要求使用随机数，因为这个小小的重放弱点会使未随机化的摘要认证变得和基本认证一样脆弱。随机数是在 `WWW-Authenticate` 质询中从服务器传送给客户端的。

13.1.4 摘要认证的握手机制

HTTP 摘要认证协议是一种升级版的认证方式，所用首部与基本认证类似。它在传统首部中添加了一些新的选项，还添加了一个新的可选首部 `Authorization-Info`。

图 13-2 描述了简化的摘要认证三步握手机制。

图 13-2 中发生的情况如下所述。

- 在第（1）步中，服务器会计算出一个随机数。在第（2）步中，服务器将这个随机数放在 `WWW-Authenticate` 质询报文中，与服务器所支持的算法列表一同发往客户端。 290
- 在第（3）步中，客户端选择一个算法，计算出密码和其他数据的摘要。在第（4）步中，将摘要放在一条 `Authorization` 报文中发回服务器。如果客户端要对服务器进行认证，可以发送客户端随机数。
- 在第（5）步中，服务器接收摘要、选中的算法以及支撑数据，计算出与客户端相同的摘要。然后服务器将本地生成的摘要与网络传送过来的摘要进行比较，验

注 9：随机数这个词表示“本次”或“临时的”。在计算机安全的概念中，随机数捕获了一个特定的时间点，将其加入到安全计算之中。

证其是否匹配。如果客户端反过来用客户端随机数对服务器进行质询，就会创建客户端摘要。服务器可以预先将下一个随机数计算出来，提前将其传递给客户端，这样下一次客户端就可以预先发送正确的摘要了。

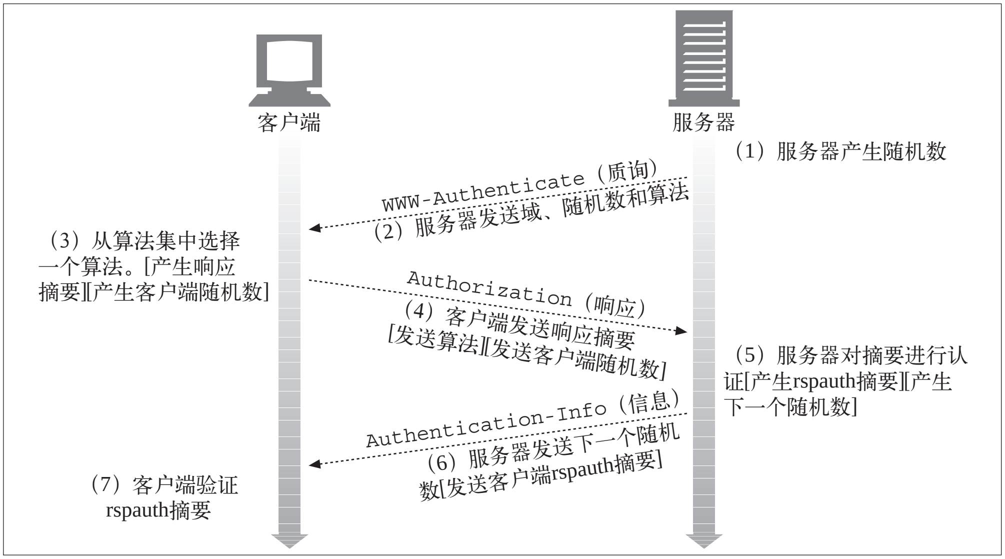


图 13-2　摘要认证的握手机制

这些信息中很多是可选的，而且有默认值。为了说明问题，图 13-3 对比了基本认证中发送的报文（参见图 13-3a 至图 13-3d）与简单的摘要认证实例发送的报文（参见图 13-3e 至图 13-3h）。

现在我们来更详细地探讨摘要认证的内部工作原理。

13.2　摘要的计算

摘要认证的核心就是对公共信息、保密信息和有时限的随机值这个组合的单向摘要。现在我们来看看这些摘要是如何计算出来的。摘要计算通常都是简单易懂的。[10] 附录 F 提供了示例源代码。

13.2.1　摘要算法的输入数据

摘要是根据以下三个组件计算出来的。

- 由单向散列函数 H(*d*) 和摘要 KD(*s*,*d*) 组成的一对函数，其中 *s* 表示密码，*d* 表示数据。

注 10：但对初学者来说，可选的 RFC 2617 兼容模式以及规范中背景资料的缺乏，使其变得有些复杂。我们会努力提供一些帮助。

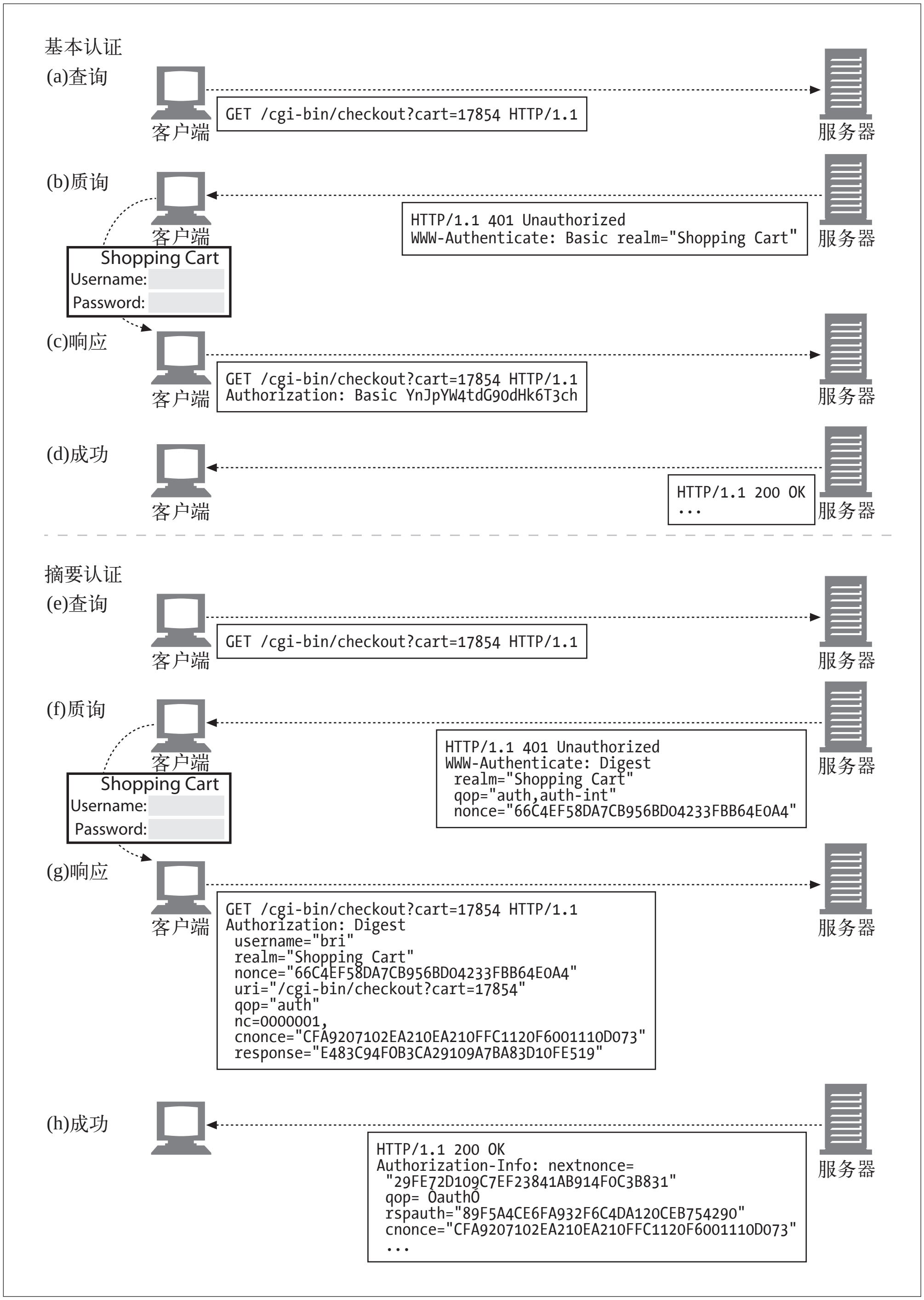


图 13-3　基本认证与摘要认证的语法对比

- 一个包含了安全信息的数据块，包括密码，称为 A1。
- 一个包含了请求报文中非保密属性的数据块，称为 A2。

H 和 KD 处理两块数据 A1 和 A2，产生摘要。

13.2.2　算法H(*d*)和KD(*s*,*d*)

291～292 摘要认证支持对各种摘要算法的选择。RFC 2617 建议的两种算法为 MD5 和 MD5-sess（“sess”表示会话），如果没有指定其他算法，默认算法为 MD5。

不管使用的是 MD5 还是 MD5-sess，都会用函数 H 来计算数据的 MD5，用摘要函数 KD 来计算以冒号连接的密码和非保密数据的 MD5。例如：

```
H(<data>) = MD5(<data>)
KD(<secret>,<data>) = H(concatenate(<secret>:<data>))
```

13.2.3　与安全性相关的数据（A1）

被称为 A1 的数据块是密码和受保护信息的产物，它包含有用户名、密码、保护域和随机数等内容。A1 只涉及安全信息，与底层报文自身无关。A1 会与 H、KD 和 A2 一同用于摘要计算。

RFC 2617 根据选择的算法定义了两种计算 A1 的方式。

- MD5

为每条请求运行单向散列函数。A1 是由冒号连接起来的用户名、域以及密码三元组。

- MD5-sess

只在第一次 `WWW-Authenticate` 握手时运行一次散列函数。对用户名、域和密码进行一次 CPU 密集型散列，并将其放在当前随机数和客户端随机数（cnonce）的前面。

表 13-2 显示了 A1 的定义。

表13-2　算法对A1的定义

算法	A1
MD5	`A1 = <user>:<realm>:<password>`
MD5-sess	`A1 = MD5(<user>:<realm>:<password>):<nonce>:<cnonce>`

13.2.4　与报文有关的数据（A2）

数据块 A2 表示的是与报文自身有关的信息，比如 URL、请求方法和报文实体的主

体部分。A2 有助于防止方法、资源或报文被篡改。A2 会与 H、KD 和 A1 一起用于摘要的计算。

RFC 2617 根据所选择的保护质量（qop），为 A2 定义了两种策略。

- 第一种策略只包含 HTTP 请求方法和 URL。当 qop="auth" 时使用这种策略，这是默认的情况。
- 第二种策略添加了报文实体的主体部分，以提供一定程度的报文完整性检测。qop="auth-int" 时使用。 293

表 13-3 显示了 A2 的定义。

表13-3　算法对A2的定义（请求摘要）

qop	A2
未定义	<request-method>:<uri-directive-value>
auth	<request-method>:<uri-directive-value>
auth-int	<request-method>:<uri-directive-value>:H(<request-entity-body>)

request-method 是 HTTP 的请求方法。uri-directive-value 是请求行中的请求 URI。可能是个 "*"、absoluteURL 或者 abs_path，但它必须与请求 URI 一致。尤其需要注意的是，如果请求 URI 是 absoluteURL，它必须是个绝对 URL。

13.2.5　摘要算法总述

RFC 2617 定义了两种给定了 H、KD、A1 和 A2 之后，计算摘要的方式。

- 第一种方式要与老规范 RFC 2069 兼容，在没有 qop 选项的时候使用。它是用保密信息和随机报文数据的散列值来计算摘要的。
- 第二种方式是现在推荐使用的方式——这种方式包含了对随机数计算和对称认证的支持。只要 qop 为 auth 或 auth-int，就要使用这种方式。它向摘要中添加了随机计数、qop 和 cnonce 数据。

表 13-4 给出了得到的摘要函数定义。注意得到的摘要使用了 H、KD、A1 和 A2。

表13-4　新/老摘要算法

qop	摘要算法	备　注
未定义	KD(H(A1), <nonce>:H(A2))	不推荐
auth 或 auth-int	KD(H(A1), <nonce>:<nc>:<cnonce>:<qop>:H(A2))	推荐

这些派生封装层很容易把人弄晕。这也是有些读者觉得 RFC 2617 难懂的原因之一。为了简化，表 13-5 扩展了 H 和 KD 的定义，用 A1 和 A2 来表示摘要。

表13-5　展开的摘要算法备忘单

294

qop	算　法	展开的算法
未定义	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:MD5(A2))
auth	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:<nc>:<cnonce>:<qop>:MD5(A2))
auth-int	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:<nc>:<cnonce>:<qop>:MD5(A2))

13.2.6　摘要认证会话

客户端响应对保护空间的 `WWW-Authenticate` 质询时，会启动一个此保护空间的认证会话（与受访问服务器的标准根结合在一起的域就定义了一个“保护空间”）。

在客户端收到另一条来自保护空间的任意一台服务器的 `WWW-Authenticate` 质询之前，认证会话会一直持续。客户端应该记住用户名、密码、随机数、随机数计数以及一些与认证会话有关的隐晦值，以便将来在此保护空间中构建请求的 `Authorization` 首部时使用。

随机数过期时，即便老的 `Authorization` 首部所包含的随机数不再新鲜了，服务器也可以选择接受其中的信息。服务器也可以返回一个带有新随机数的 401 响应，让客户端重试这条请求；指定这个响应为 `stale=true`，表示服务器在告知客户端用新的随机数来重试，而不再重新提示输入新的用户名和密码了。

13.2.7　预授权

在普通的认证方式中，事务结束之前，每条请求都要有一次请求 / 质询的循环，参见图 13-4a。

如果客户端事先知道下一个随机数是什么，就可以取消这个请求 / 质询循环，这样客户端就可以在服务器发出请求之前，生成正确的 `Authorization` 首部了。如果客户端能在服务器要求它计算 `Authorization` 首部之前将其计算出来，就可以预先将 `Authorization` 首部发送给服务器，而不用进行请求 / 质询了。图 13-4b 显示了这种方式对性能的影响。

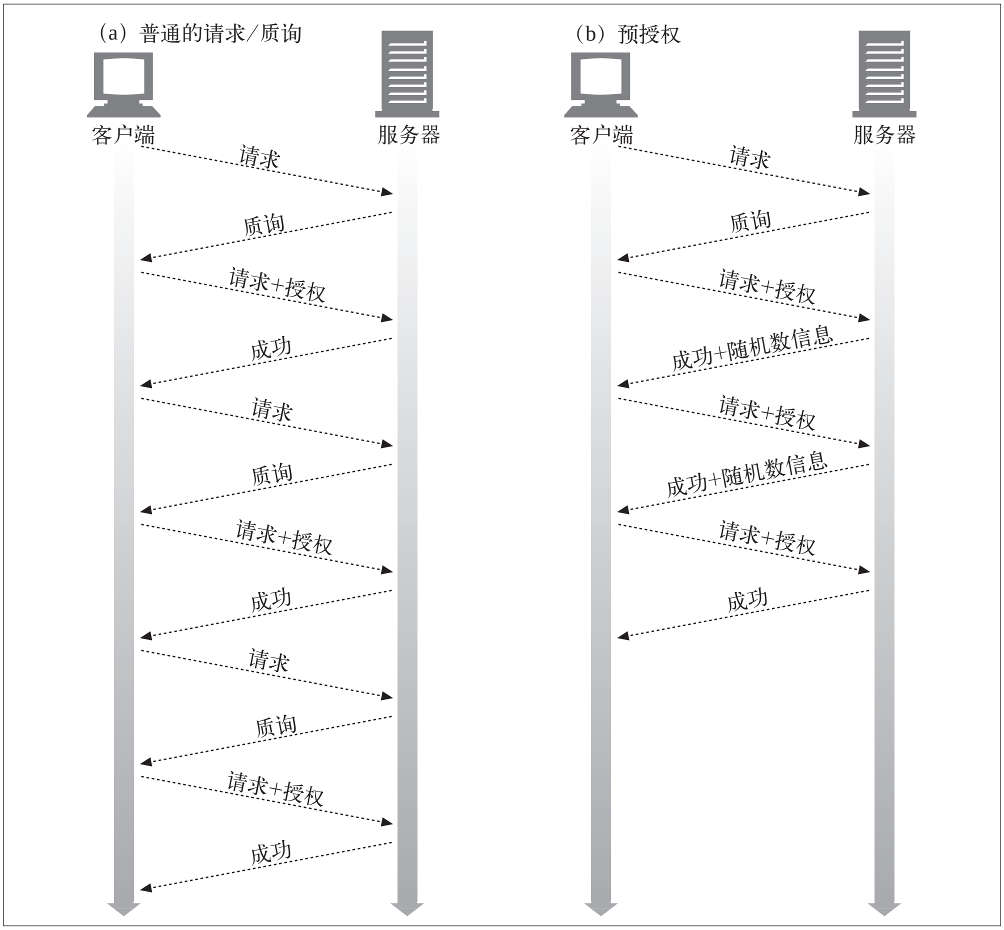


图 13-4　预授权减少了报文的数量

预授权对基本认证来说并不重要（而且很常见）。浏览器通常会维护一些客户端数据库以存储用户名和密码。一旦用户与某站点进行了认证，浏览器通常会为后继对那个 URL 的请求发送正确的 Authorization 首部（参见第 12 章）。 295

由于摘要认证使用了随机数技术来破坏重放攻击，所以对摘要认证来说，预授权要稍微复杂一些。服务器会产生任意的随机数，所以在客户端收到质询之前，不一定总能判定应该发送什么样的 Authorization 首部。

摘要认证在保留了很多安全特性的同时，还提供了几种预授权方式。这里列出了三种可选的方式，通过这些方式，客户端无需等待新的 WWW-Authenticate 质询，就可以获得正确的随机数：

- 服务器预先在 Authentication-Info 成功首部中发送下一个随机数；

- 服务器允许在一小段时间内使用同一个随机数；
- 296 客户端和服务器使用同步的、可预测的随机数生成算法。

1. 预先生成下一个随机数

可以在 Authentication-Info 成功首部中将下一个随机数预先提供给客户端。这个首部是与前一次成功认证的 200 OK 响应一同发送的。

```
Authentication-Info: nextnonce="<nonce-value>"
```

有了下一个随机数，客户端就可以预先发布 Authorization 首部了。

尽管这种预授权机制避免了请求 / 质询循环（加快了事务处理的速度），但实际上它也破坏了对同一台服务器的多条请求进行管道化的功能，因为在发布下一条请求之前，一定要收到下一个随机值才行。而管道化是避免延迟的一项基本技术，所以这样可能会造成很大的性能损失。

2. 受限的随机数重用机制

另一种方法不是预先生成随机数序列，而是在有限的次数内重用随机数。比如，服务器可能允许将某个随机数重用 5 次，或者重用 10 秒。

在这种情况下，客户端可以随意发布带有 Authorization 首部的请求，而且由于随机数是事先知道的，所以还可以对请求进行管道化。随机数过期时，服务器要向客户端发送 401 Unauthorized 质询，并设置 WWW-Authenticate:stale=true 指令：

```
WWW-Authenticate: Digest
    realm="<realm-value>"
    nonce="<nonce-value>"
    stale=true
```

重用随机数使得攻击者更容易成功地实行重放攻击。虽然这确实降低了安全性，但重用的随机数的生存期是可控的（从严格禁止重用到较长时间的重用），所以应该可以在安全和性能间找到平衡。

此外，还可以通过其他一些特性使重放攻击变得更加困难，其中就包括增量计数器和 IP 地址测试。但这些技术只能使攻击的实施更加麻烦，并不能消除由此带来的安全隐患。

3. 同步生成随机数

还可以采用时间同步的随机数生成算法，客户端和服务器可根据共享的密钥，生成第三方无法轻易预测的、相同的随机数序列（比如安全 ID 卡）。

297 这些算法都超出了摘要认证规范的范畴。

13.2.8 随机数的选择

随机数的内容不透明，而且与实现有关。但性能、安全性和便捷性的优劣都取决于明智的选择。

RFC 2617 建议采用这个假想的随机数公式：

```
BASE64(time-stamp H(time-stamp ":" ETag ":" private-key))
```

其中 `time-stamp` 是服务器产生的时间或其他不会重复的值，`ETag` 是与所请求实体有关的 HTTP `ETag` 首部的值，`private-key` 是只有服务器知道的数据。

有了这种形式的随机数，服务器就可以在收到客户端的认证首部之后重新计算散列部分，如果结果与那个首部的随机数不符，或者时间戳的值不够新，就拒绝请求。服务器可以通过这种方式来限制随机数的有效持续时间。

包含 `Etag` 可以防止对已更新资源版本的重放请求。（注意，在随机数中包含客户端的 IP 地址，服务器好像就可以限制原来获得此随机数的客户端重用这个随机数了，但这会破坏代理集群的工作。使用代理集群时，来自单个用户的多条请求通常会经过不同的代理进行传输，而且 IP 地址欺骗实现起来也不是很难。）

实现可以选择不接受以前使用过的随机数或摘要，以防止重放攻击。实现也可以选择为 POST 或 PUT 请求使用一次性的随机数或摘要，为 GET 请求使用时间戳。

会影响到随机数选取的一些实际安全问题参见 13.5 节。

13.2.9 对称认证

RFC 2617 扩展了摘要认证机制，允许客户端对服务器进行认证。这是通过提供客户端随机值来实现的，服务器会根据它对共享保密信息的正确了解生成正确的响应摘要。然后，服务器在 `Authorization-Info` 首部中将此摘要返回给客户端。

这种对称认证方式被标准化为 RFC 2617。为了与原有 RFC 2069 标准后向兼容，它是可选的，但由于它提供了一些重要的安全提升机制，强烈推荐现今所有的客户端和服务器都要实现全部 RFC 2617 特性。特别是，只要提供了 `qop` 指令，就要求执行对称认证，而没有 `qop` 指令时则不要求执行对称认证。

响应摘要的计算方法与请求摘要类似，但由于响应中没有方法，而且报文实体数据有所不同，所以只有报文主体信息 A2 不同。表 13-6 和表 13-7 对比了请求和响应摘要中 A2 的计算方法。 298

表13-6　算法中A2的定义（请求摘要）

qop	A2
未定义	<request-method>:<uri-directive-value>
auth	<request-method>:<uri-directive-value>
auth-int	<request-method>:<uri-directive-value>:H(<request-entity-body>)

表13-7　算法中A2的定义（响应摘要）

qop	A2
未定义	:<uri-directive-value>
auth	:<uri-directive-value>
auth-int	:<uri-directive-value>:H(<response-entity-body>)

cnonce 值和 nc 值必须是本报文所响应的客户端请求中的相应值。如果指定了 qop="auth" 或 qop="auth-int"，就必须提供响应 auth、cnonce 和 nonce 计数指令。

13.3　增强保护质量

可以在三种摘要首部中提供 qop 字段：WWW-Authenticate、Authorization 和 Authentication-Info。

通过 qop 字段，客户端和服务器可以对不同类型及质量的保护进行协商。比如，即便会严重降低传输速度，有些事务可能也要检查报文主体的完整性。

服务器首先在 WWW-Authenticate 首部输出由逗号分隔的 qop 选项列表。然后客户端从中选择一个它支持且满足其需求的选项，并将其放在 Authorization 的 qop 字段中回送给服务器。

qop 字段是可选的，但只是在后向兼容原有 RFC 2069 规范的情况下才是可选的。现代所有的摘要实现都应该支持 qop 选项。

RFC 2617 定义了两种保护质量的初始值：表示认证的 auth，带有报文完整性保护的认证 auth-int。将来可能还会出现其他 qop 选项。

13.3.1　报文完整性保护

如果使用了完整性保护（qop="auth-int"），H（实体的主体部分）就是对实体主体部分，而不是报文主体部分的散列。对于发送者，要在应用任意传输编码方式之前计算；而对于接收者，则应在去除所有传输编码之后计算。注意，对于任何含有
299 多部份的内容类型来说，多部分的边界和每部分中嵌入的首部都要包含在内。

13.3.2 摘要认证首部

基本认证和摘要认证协议都包含了 WWW-Authenticate 首部承载的授权质询、Authorization 首部承载的授权响应。摘要认证还添加了可选的 Authorization-Info 首部，这个首部是在成功认证之后发送的，用于实现三步握手机制，并传送下一个随机数。表 13-8 给出了基本认证和摘要认证的首部。

表13-8　HTTP认证首部

阶段	基　本	摘　要
质询	WWW-Authenticate: Basic realm="<realm-value>"	WWW-Authenticate: Digest realm="<realm-value>" nonce="<nonce-value>" [domain="<list-of-URIs>"] [opaque="<opaque-token-value>"] [stale=<true-or-false>] [algorithm=<digest-algorithm>] [qop="<list-of-qop-values>"] [<extension-directive>]
响应	Authorization: Basic <base64(user:pass)>	Authorization: Digest username="<username>" realm="<realm-value>" nonce="<nonce-value>" uri=<request-uri> response="<32-hex-digit-digest>" [algorithm=<digest-algorithm>] [opaque="<opaque-token-value>"] [cnonce="<nonce-value>"] [qop=<qop-value>] [nc=<8-hex-digit-nonce-count>] [<extension-directive>]
Info	n/a	Authentication-Info: nextnonce="<nonce-value>" [qop="<list-of-qop-values>"] [rspauth="<hex-digest>"] [cnonce="<nonce-value>"] [nc=<8-hex-digit-nonce-count>]

摘要认证首部要复杂得多。详细介绍参见附录 F。

13.4 应该考虑的实际问题

使用摘要认证时需要考虑几件事情。本节讨论了其中一些问题。 300

13.4.1 多重质询

服务器可以对某个资源发起多重质询。比如，如果服务器不了解客户端的能力，就可以既提供基本认证质询，又提供摘要认证质询。客户端面对多重质询时，必须以它所支持的最强的质询机制来应答。

质询自身可能会包含由逗号分隔的认证参数列表。如果 `WWW-Authenticate` 或 `Proxy-Authenticate` 首部包含了多个质询，或者提供了多个 `WWW-Authenticate` 首部，用户 Agent 代理在解析 `WWW-Authenticate` 或 `Proxy-Authenticate` 首部字段值时就要特别小心。注意，很多浏览器只支持基本认证，要求这是提交给它的第一种认证机制。

在提供了认证选项范围的情况下，安全问题上就会存在明显的“最薄弱环节”。只有当基本认证是最低可接受认证方式时，服务器才应该包含它，而且管理员还应该警告用户，即使运行了不同层次安全措施，系统间使用相同密码也存在一定危险性。

13.4.2 差错处理

在摘要认证中，如果某个指令或其值使用不当，或者缺少某个必要指令，就应该使用响应 400 Bad Request。

如果请求的摘要不匹配，就应该记录一次登录失败。某客户端连续多次失败可能说明有攻击者正在猜测密码。

认证服务器一定要确保 URI 指令指定的资源与请求行中指定的资源相同。如果不同，服务器就应该返回 400 Bad Request 错误。（这可能是一种攻击的迹象，因此服务器设计者可能会考虑将此类错误记录下来。）这个字段包含的内容与请求 URL 中的内容是重复的，用来应对中间代理可能对客户端请求进行的修改。这个经过修改（但估计语义是等价的）的请求计算后得到的摘要可能会与客户端计算出的摘要有所不同。

13.4.3 保护空间

域值，与被访问服务器的标准根 URL 结合在一起，定义了保护空间。

通过域可以将服务器上的受保护资源划分为一组保护空间，每个空间都有自己的认证机制和 / 或授权数据库。域值是一个字符串，通常由原始服务器分配，可能会有认证方案特有的附加语义。注意，可能会有多个授权方案相同，而域不同的质询。

301

保护空间确定了可以自动应用证书的区域。如果前面的某条请求已被授权，在一段

时间内，该保护空间中所有其他请求都可以重用同一个证书，时间的长短由认证方案、参数和 / 或用户喜好来决定。除非认证方案进行了其他定义，否则单个保护空间是不能扩展到其服务器范围之外的。

对保护空间的具体计算取决于认证机制。

- 在基本认证中，客户端会假定请求 URI 中或其下的所有路径都与当前的质询处于同一个保护空间内。客户端可以预先提交对此空间中资源的认证，无需等待来自服务器的另一条质询。
- 在摘要认证中，质询的 `WWW-Authenticate:domain` 字段对保护空间作了更精确的定义。`domain` 字段是一个用引号括起来的、中间由空格分隔的 URI 列表。通常认为，`domain` 列表中的所有 URI 和逻辑上处于这些前缀之下的所有 URI，都位于同一个保护空间中。如果没有 `domain` 字段，或者此字段为空，质询服务器上的所有 URI 就都在保护空间内。

13.4.4 重写URI

代理可以通过改变 URI 语法，而不改变所描述的实际资源的方式来重写 URI。比如：

- 可以对主机名进行标准化，或用 IP 地址来取代；
- 可以用“%”转义形式来取代嵌入的字符；
- 如果某类型的一些附加属性不会影响从特定原始服务器上获取资源，就可以将其附加或插入到 URI 中。

代理可修改 URI，而且摘要认证会检查 URI 值的完整性，所以如果进行了任意一种修改，摘要认证就会被破坏。更多信息参见 13.2.4 节。

13.4.5 缓存

共享的缓存收到包含 `Authorization` 首部的请求和转接那条请求产生的响应时，除非响应中提供了下列两种 `Cache-Control` 指令之一，否则一定不能将那条响应作为对任何其他请求的应答使用。

- 如果原始响应中包含有 `Cache-Control` 指令 `must-revalidate`，缓存可以在应答后继请求时使用那条响应的实体部分。但它首先要用新请求的请求首部，与原始服务器再次进行验证，这样原始服务器就可以对新请求进行认证了。
- 如果原始响应中包含有 `Cache-Control` 指令 `public`，在对任意后继请求的应答中都可以返回响应的实体部分。 302

13.5 安全性考虑

RFC 2617 总结了 HTTP 认证方案固有的一些安全风险，这是很令人钦佩的。本节描述了其中的部分风险。

13.5.1 首部篡改

为了提供一个简单明了的防首部篡改系统，要么就得进行端到端的加密，要么就得对首部进行数字签名——最好是两者的结合！摘要认证的重点在于提供一种防篡改认证机制，但并不一定要将这种保护扩展到数据上去。具有一定保护级别的首部只有 `WWW-Authenticate` 和 `Authorization`。

13.5.2 重放攻击

在当前的上下文中，重放攻击指的就是有人将从某个事务中窃取的认证证书用于另一个事务。尽管对 GET 请求来说这也是个问题，但为 POST 和 PUT 请求提供一种简单的方式来避免重放攻击才是非常必要的。在传输表单数据的同时，成功重放原先用过的证书会引发严重的安全问题。

因此，为了使服务器能够接受“重放的”证书，还必须重复发送随机数。缓解这个问题的方法之一就是让服务器产生的随机数包含（如前所述）根据客户端 IP 地址、时间戳、资源 Etag 和私有服务器密钥算出的摘要。这样，IP 地址和一个短小超时值的组合就会给攻击者造成很大的障碍。

但这种解决方案有一个很重要的缺陷。我们之前讨论过，用客户端 IP 地址来创建随机数会破坏经过代理集群的传输。在这类传输中。来自单个用户的多条请求可能会穿过不同的代理。而且，IP 欺骗也并不难实现。

一种可以完全避免重放攻击的方法就是为每个事务都使用一个唯一的随机数。在这种实现方式中，服务器会为每个事务发布唯一的随机数和一个超时值。发布的随机数只对指定的事务有效，而且只在超时值的持续区间内有效。这种方式会增加服务器的负担，但这种负担可忽略不计。

13.5.3 多重认证机制

服务器支持多重认证机制（比如基本认证和摘要认证）时，通常会在 `WWW-Authenticate`
303 首部提供选项。由于没有要求客户端选择功能最强的认证机制，所以得到的认证效果就和功能最弱的认证方案差不多。

要避免出现这个问题，最直接的方法就是让客户端总是去选择可用认证方案中功能

最强的那个。如果无法实现（因为大部分人使用的都是商业化客户端），唯一的选择就是使用一个只维护最强认证方案的代理服务器。但只有在已知所有客户端都支持所选认证方案的区域中才能采用这种方式——比如，在公司网络中。

13.5.4　词典攻击

词典攻击是典型的密码猜测型攻击方式。恶意用户对某个事务进行窃听，并对随机数 / 响应对使用标准的密码猜测程序。如果用户使用的是相对比较简单的密码，而且服务器使用的也是简单的随机数，它很可能会找到匹配项。如果没有密码过期策略，只要有足够的时间和破解密码所需的一次性费用，就很容易搜集到足够多的密码，造成实质性的破坏。

除了使用复杂的相对难以破译的密码和合适的密码过期策略之外，确实没有什么好的方法可以解决这个问题。

13.5.5　恶意代理攻击和中间人攻击

现在很多因特网流量都会在这个或那个地方流经某个代理。随着重定向技术和拦截代理的出现，用户甚至都意识不到他的请求穿过了某个代理。如果这些代理中有一个是恶意的或者容易被入侵的，就会使客户端置于中间人攻击之下。

这种攻击可以采用窃听的形式，也可以删除提供的所有选项，用最薄弱的认证策略（比如基本认证）来取代现有的认证机制，对其进行修改。

入侵受信代理的方式之一就是使用其扩展接口。有时代理会提供复杂的编程接口，可以为这类代理编写一个扩展（比如，plug-in）来拦截流量并对其进行修改。不过，数据中心和代理自身提供的安全性使得通过恶意 plug-in 进行中间人攻击的可能性变得很渺茫。

没有什么好办法可以解决这个问题。可行的解决方案包括由客户端提供与认证功能有关的可见线索，对客户端进行配置使其总是使用可用认证策略中功能最强的那一种，等等。但即使使用的是最强大的认证策略，客户端仍然很容易被窃听。防止这些攻击唯一简便的方式就是使用 SSL。 304

13.5.6　选择明文攻击

使用摘要认证的客户端会用服务器提供的随机数来生成响应。但如果中间有一个被入侵的或恶意的代理在拦截流量（或者有个恶意的原始服务器），就可以很容易地为客户端的响应计算提供随机数。使用已知密钥来计算响应可以简化响应的密码分析过程。这种方式被称为*选择明文攻击*（chosen plaintext attack）。选择明文攻击有以下几种变体形式。

- *预先计算的词典攻击*

  这是词典攻击和选择明文攻击的组合。首先，发起攻击的服务器会用预先确定的随机数和常见密码的变化形式产生一组响应，创建一个词典。一旦有了规模可观的词典，攻击服务器或代理就可以完成对流量的封锁，向客户端发送预先确定的随机数。攻击者从客户端得到一个响应时，会搜索生成的词典，寻找匹配项。如果有匹配项，攻击者就捕获了这个用户的密码。

- *批量暴力型攻击*

  批量暴力型攻击的不同之处在于计算密码的方式。它没有试图去匹配预先计算出来的摘要，而是用一组机器枚举了指定空间内所有可能的密码。随着机器运行速度变得越来越快，暴力型攻击的可行性也变得越来越强了。

总之，这些攻击所造成的威胁是很容易应对的。防止这些攻击的一种方法就是配置客户端使用可选的 `cnonce` 指令，这样响应就是基于客户端的判断产生的，而不是用服务器提供的随机数（这个随机数可能会被攻击者入侵）产生的。通过这种方法，再结合一些强制使用合理强密码的策略，以及一个好的密码过期策略，就可以完全消除选择明文攻击的威胁。

13.5.7 存储密码

摘要认证机制将对比用户的响应与服务器内部存储的内容——通常就是用户名和 H(A1) 元组对，其中 H(A1) 是从用户名、域和密码的摘要中导出的。

与 Unix 机器中传统的密码文件不同，如果摘要认证密码文件被入侵了，攻击者马上就能够使用域中所有文件，不需要再进行解码了。

消除这个问题的方法包括：

- 就像密码文件中包含的是明文密码一样来保护它；
- 确保域名在所有域中是唯一的。这样，如果密码文件被入侵，所造成的破坏也只
305 局限于一个特定的域中。包含主机和 `domain` 的全路径域名就可以满足这个要求。

尽管摘要认证提供的解决方案比基本认证要强壮且安全得多，但它并没有为内容的安全提供任何保证——真正安全的事务只有通过 SSL 才能实现，我们将在下一章介绍。

13.6 更多信息

更多有关认证的信息，参见以下资源。

- http://www.ietf.org/rfc/rfc2617.txt

RFC 2617，“HTTP Authentication：Basic and Digest Access Authentication.”
306 （“HTTP 认证：基本和摘要访问认证”）。

第14章

安全HTTP

前面三章讨论了一些有助于识别和认证用户的 HTTP 特性。在友好环境中，这些技术都能够很好地工作，但在充满各种利益驱动和恶意对手的环境中，它们并不足以保护那些重要的事务处理。

本章提供了一种更复杂，更安全的技术，通过数字密码来保护 HTTP 事务免受窃听和篡改的侵害。

14.1 保护HTTP的安全

人们会用 Web 事务来处理一些很重要的事情。如果没有强有力的安全保证，人们就无法安心地进行网络购物或使用银行业务。如果无法严格限制访问权限，公司就不能将重要的文档放在 Web 服务器上。Web 需要一种安全的 HTTP 形式。

前面的章节讨论了一些提供认证（基本认证和摘要认证）和报文完整性检查（摘要 `qop="auth-int"`）的轻量级方法。对很多网络事务来说，这些方法都是很好用的，但对大规模的购物、银行事务，或者对访问机密数据来说，并不足够强大。这些更为重要的事务需要将 HTTP 和数字加密技术结合起来使用，才能确保安全。

HTTP 的安全版本要高效、可移植且易于管理，不但能够适应不断变化的情况而且还应该能满足社会和政府的各项要求。我们需要一种能够提供下列功能的 HTTP 安全技术。

- 服务器认证（客户端知道它们是在与真正的而不是伪造的服务器通话）。
- 客户端认证（服务器知道它们是在与真正的而不是伪造的客户端通话）。
- 完整性（客户端和服务器的数据不会被修改）。
- 加密（客户端和服务器的对话是私密的，无需担心被窃听）。
- 效率（一个运行的足够快的算法，以便低端的客户端和服务器使用）。
307 - 普适性（基本上所有的客户端和服务器都支持这些协议）。
- 管理的可扩展性（在任何地方的任何人都可以立即进行安全通信）。
- 适应性（能够支持当前最知名的安全方法）。
- 在社会上的可行性（满足社会的政治文化需要）。

HTTPS

HTTPS 是最流行的 HTTP 安全形式。它是由网景公司首创的，所有主要的浏览器和服务器都支持此协议。

HTTPS 方案的 URL 以 https://，而不是 http:// 开头，据此就可以分辨某个 Web 页面是通过 HTTPS 而不是 HTTP 访问的（有些浏览器还会显示一些标志性的安全提示，如图 14-1 所示）。

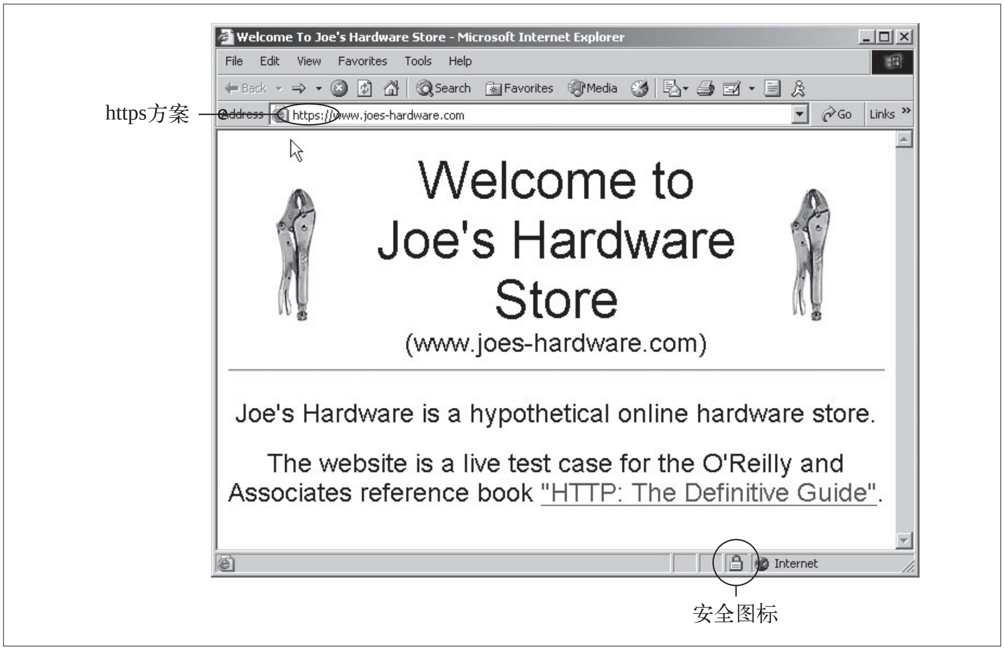


图 14-1　浏览安全 Web 站点

使用 HTTPS 时，所有的 HTTP 请求和响应数据在发送到网络之前，都要进行加密。HTTPS 在 HTTP 下面提供了一个传输级的密码安全层（参见图 14-2）——可以使用 SSL，也可以使用其后继者——传输层安全（Transport Layer Security，TLS）。由于 SSL 和 TLS 非常类似，所以在本书中我们不太严格地用术语 SSL 来表示 SSL 和 TLS。

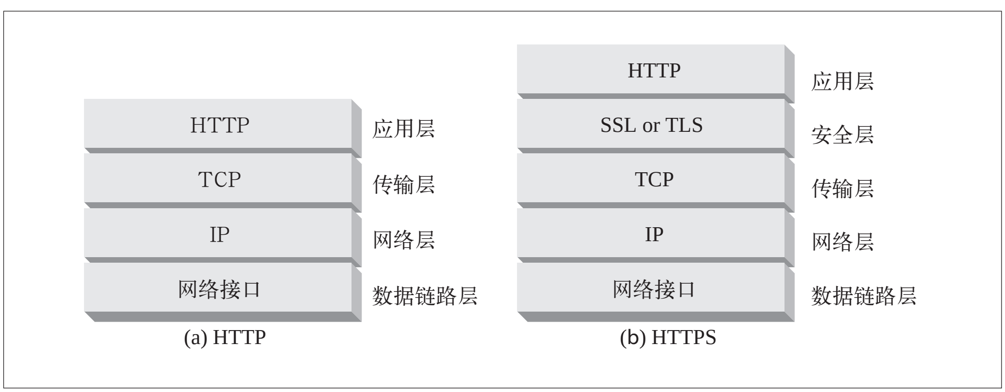


图 14-2　HTTPS 是位于安全层之上的 HTTP，这个安全层位于 TCP 之上

大部分困难的编码及解码工作都是在 SSL 库中完成的，所以 Web 客户端和服务器在使用安全 HTTP 时无需过多地修改其协议处理逻辑。在大多数情况下，只需要用 308

SSL 的输入 / 输出调用取代 TCP 的调用，再增加其他几个调用来配置和管理安全信息就行了。

14.2　数字加密

在详细探讨 HTTPS 之前，我们先介绍一些 SSL 和 HTTPS 用到的加密编码技术的背景知识。在接下来的几节里，我们会对数字加密的本质进行一个快速的入门性介绍。如果你已经掌握了数字加密的技术和术语，可以直接阅读 14.7 节。

在这个数字加密技术的入门介绍中，我们会讨论以下内容。

- *密码*

对文本进行编码，使偷窥者无法识别的算法。

- *密钥*

改变密码行为的数字化参数。

- *对称密钥加密系统*

编 / 解码使用相同密钥的算法。

- *不对称密钥加密系统*

编 / 解码使用不同密钥的算法。

- *公开密钥加密系统*

一种能够使数百万计算机便捷地发送机密报文的系统。

- *数字签名*

用来验证报文未被伪造或篡改的校验和。

- *数字证书*

309 由一个可信的组织验证和签发的识别信息。

14.2.1　密码编制的机制与技巧

密码学是对报文进行编 / 解码的机制与技巧。人们用加密的方式来发送秘密信息已经有数千年了。但密码学所能做的还不仅仅是加密报文以防止好事者的读取，我们还可以用它来防止对报文的篡改，甚至还可以用密码学来证明某条报文或某个事务确实出自你手，就像支票上的手写签名或信封上的压纹封蜡一样。

14.2.2 密码

密码学基于一种名为*密码*（cipher）的秘密代码。密码是一套编码方案——一种特殊的报文编码方式和一种稍后使用的相应解码方式的结合体。加密之前的原始报文通常被称为*明文*（plaintext 或 cleartext）。使用了密码之后的编码报文通常被称作*密文*（ciphertext）。图 14-3 显示了一个简单的例子。

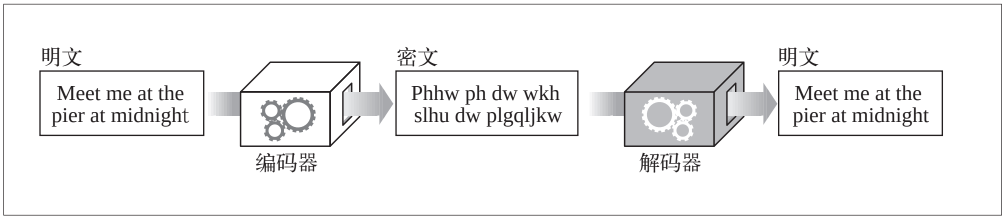


图 14-3　明文和密文

用密码来生成保密信息已经有数千年了。传说尤利乌斯 • 凯撒（Julius Caesar）曾使用过一种三字符循环移位密码，报文中的每个字符都由字母表中三个位置之后的字符来取代。在现代的字母表中，“A”就应该由“D”来取代，“B”就应该由“E”来取代，以此类推。

比如，在图 14-4 中，用 rot3（循环移位 3 字符）密码就可以将报文“meet me at the pier at midnight”编码为密文“phhw ph dw wkh slhu dw plgqljkw”。[1] 通过解码，在字母表中循环移位 3 个字符，就可以将密文解密回原来的明文报文。

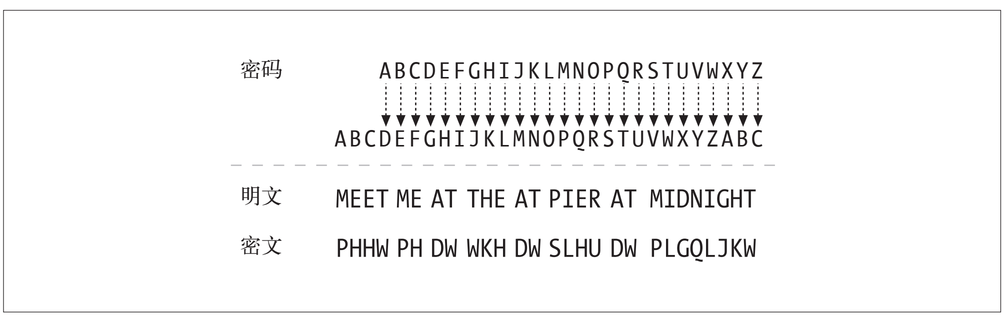


图 14-4　循环移位 3 字符密码实例 310

注 1：为了简化这个例子，我们没有对标点和空格进行循环移位，但你可以自己试一试。

14.2.3 密码机

最初，人们需要自己进行编码和解码，所以起初密码是相当简单的算法。因为密码很简单，所以人们通过纸笔和密码书就可以进行编解码了，但聪明人也可以相当容易地“破解”这些密码。

随着技术的进步，人们开始制造一些机器，这些机器可以用复杂得多的密码来快速、精确地对报文进行编解码。这些密码机不仅能做一些简单的旋转，它们还可以替换字符、改变字符顺序，将报文切片切块，使代码的破解更加困难。[2]

14.2.4 使用了密钥的密码

编码算法和编码机都可能会落入敌人的手中，所以大部分机器上都有一些号盘，可以将其设置为大量不同的值以改变密码的工作方式。即使机器被盗，没有正确的号盘设置（密钥值），解码器也无法工作。[3]

这些密码参数被称为**密钥**（key）。要在密码机中输入正确的密钥，解密过程才能正确进行。密码密钥会让一个密码机看起来好像是多个虚拟密码机一样，每个密码机都有不同的密钥值，因此其行为都会有所不同。

图 14-5 显示了使用密钥的密码实例。加密算法就是普通的“循环移位 *N* 字符”密码。*N* 的值由密钥控制。将同一条输入报文“meet me at the pier at midnight”通过同一台编码机进行传输，会随密钥值的不同产生不同的输出。现在，基本上所有的加密算法都会使用密钥。

14.2.5 数字密码

随着数字计算的出现，出现了以下两个主要的进展。

311
- 从机械设备的速度和功能限制中解放出来，使复杂的编 / 解码算法成为可能。
- 支持超大密钥成为可能，这样就可以从一个加密算法中产生出数万亿的虚拟加密算法，由不同的密钥值来区分不同的算法。密钥越长，编码组合就越多，通过随机猜测密钥来破解代码就越困难。

注 2：最著名的机械编码机可能就是第二次世界大战期间德国的 Enigma 编码机了。尽管 Enigma 密码非常复杂，但阿兰 · 图灵（Alan Turing）和他的同事们在 20 世纪 40 年代初期就可以用最早的数字计算机破解 Enigma 代码了。

注 3：在现实中，机器逻辑可能会指向一些可利用的模式，所以拥有机器逻辑有时会有助于密码的破解。现代的加密算法通常都设计为，即使大家都知道这些算法，恶意的攻击者也很难发现任何有助于破解代码的模式。实际上，很多功能最强大的密码都会将其源代码放在公共域中，供大家浏览和学习！

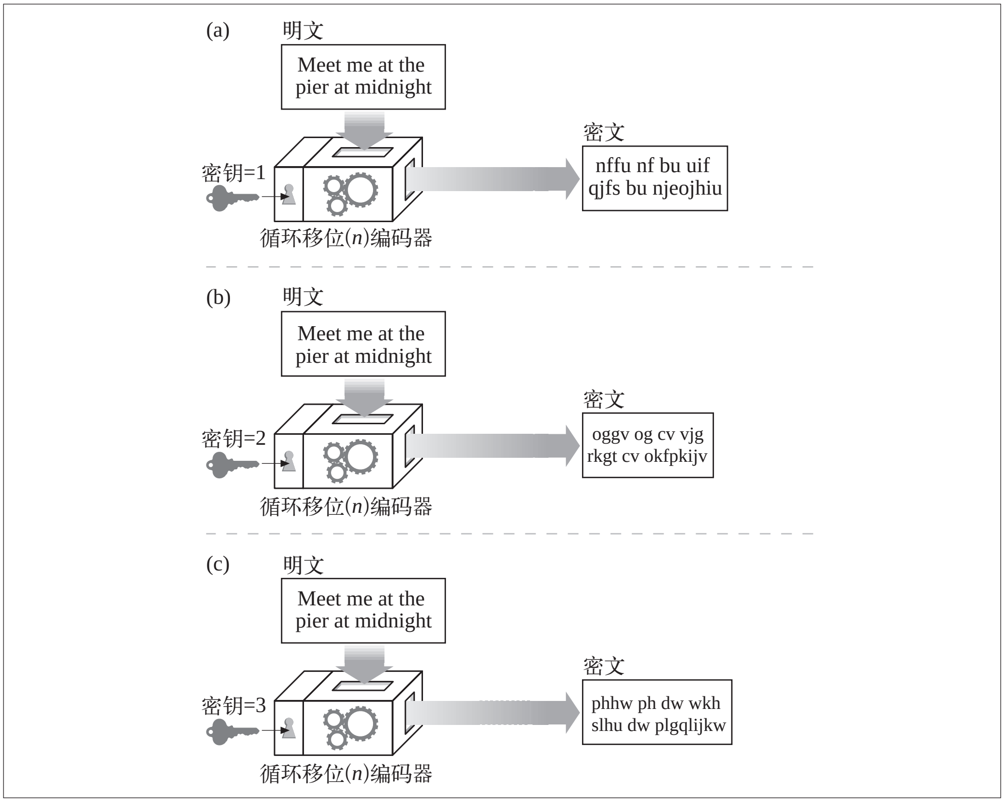


图 14-5　使用不同密钥的旋转 *N* 字符密码

与金属钥匙或机械设备中的号盘设置相比，数字密钥只是一些数字。这些数字密钥值是编 / 解码算法的输入。编码算法就是一些函数，这些函数会读取一块数据，并根据算法和密钥值对其进行编 / 解码。

给定一段明文报文 P、一个编码函数 E 和一个数字编码密钥 e，就可以生成一段经过编码的密文 C（参见图 14-6）。通过解码函数 D 和解码密钥 d，可以将密文 C 解码为原始的明文 P。当然，编 / 解码函数都是互为反函数的，对 P 的编码进行解码就会回到原始报文 P 上去。 312

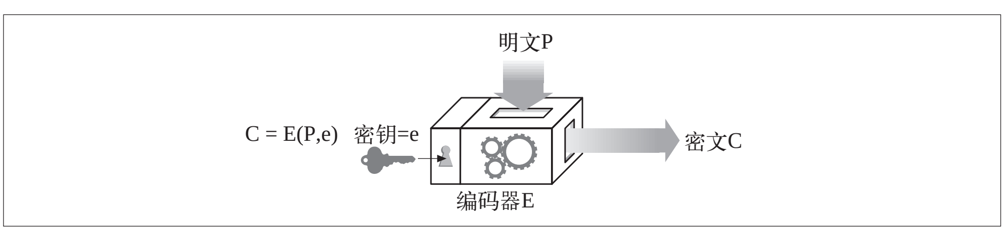


图 14-6　用编码密钥 e 对明文进行编码，用解码密钥 d 进行解码

14.3 对称密钥加密技术

我们来更详细地看看密钥和密码是怎样配合工作的。很多数字加密算法都被称为对称密钥（symmetric-key）加密技术，这是因为它们在编码时使用的密钥值和解码时一样（e=d）。我们就将其统称为密钥 k。

在对称密钥加密技术中，发送端和接收端要共享相同的密钥 k 才能进行通信。发送端用共享的密钥来加密报文，并将得到的密文发送给接收端。接收端收到密文，并对其应用解密函数和相同的共享密钥，恢复出原始的明文（参见图 14-7）。

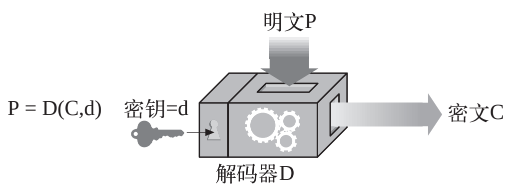


图 14-7 对称密钥加密算法为编 / 解码使用相同的密钥

流行的对称密钥加密算法包括：DES、Triple-DES、RC2 和 RC4。

14.3.1 密钥长度与枚举攻击

保持密钥的机密状态是很重要的。在很多情况下，编 / 解码算法都是众所周知的，因此密钥就是唯一保密的东西了。

好的加密算法会迫使攻击者试遍每一个可能的密钥，才能破解代码。用暴力去尝试所有的密钥值称为*枚举攻击*（enumeration attack）。如果只有几种可能的密钥值，居心不良的人通过暴力遍历所有值，就能最终破解代码了。但如果有大量可能的密钥值，他可能就要花费数天、数年，甚至无限长的时间来遍历所有的密钥，去查找能
313 够破解密码的那一个。

可用密钥值的数量取决于密钥中的位数，以及可能的密钥中有多少是有效的。就对称密钥加密技术来说，通常所有的密钥值都是有效的。[4] 8 位的密钥只有 256 个可能的密钥值，40 位的密钥可以有 2^{40} 个可能的密钥值（大约是一万亿个密钥），128 位的密钥可以产生大约 340 000 000 000 000 000 000 000 000 000 000 000 000 个可能的密钥值。

注 4：有些加密技术中只有部分密钥值是有效的。比如，在最知名的非对称密钥加密系统 RSA 中，有效密钥必须以某种方式与质数相关。可能的密钥值中只有少量密钥具备此特性。

在传统的对称密钥加密技术中，对小型的、不太重要的事务来说，40 位的密钥就足够安全了。但现在的高速工作站就可以将其破解，这些工作站每秒可以进行数十亿次计算。

相比之下，对于对称密钥加密技术，128 位的密钥被认为是非常强大的。实际上，长密钥对密码安全有着非常重要的影响，美国政府甚至对使用长密钥的加密软件实施了出口控制，以防止潜在的敌对组织创建出美国国家安全局（National Security Agency，NSA）自己都无法破解的秘密代码。

Bruce Schneier 编写的 *Applied Cryptography*（John Wiley & Sons 出版社）是一本很棒的书，书中有一张表，表中对使用 1995 年的技术和耗费，通过猜测所有的密钥来破解一个 DES 密码所需的时间进行了描述。[5] 表 14-1 摘录了这张表。

表14-1　较长的密钥要花费更多的精力去破解（来自*Applied Cryptography*一书，1995年的数据）

攻击耗费	40位密钥	56位密钥	64位密钥	80位密钥	128位密钥
100 000 美元	2 秒	35 小时	1 年	70 000 年	10^{19} 年
1 000 000 美元	200 毫秒	3.5 小时	37 天	7 000 年	10^{18} 年
10 000 000 美元	20 毫秒	21 分钟	4 天	700 年	10^{17} 年
100 000 000 美元	2 毫秒	2 分钟	9 小时	70 年	10^{16} 年
1 000 000 000 美元	200 微秒	13 秒	1 小时	7 年	10^{15} 年

根据 1995 年微处理器的速度，愿意花费 100 000 美元的攻击者可以在大约 2 秒内破解一个 40 位的 DES 代码。2002 年的计算机就已经比 1995 年的快 20 倍了。除非用户经常修改密钥，否则对于别有用心的攻击者来说，40 位的密钥是不安全的。

DES 的 56 位标准密钥长度就更安全一些。从 1995 年的经济水平来说，花费 100 万美元进行的攻击还是要几个小时才能破解密码。但可使用超级计算机的用户则只需 314
数秒钟即可通过暴力方法破解密码。与之相对的是，通常大家都认为长度与 Triple-DES 密钥相当的 128 位 DES 密钥实际上是任何人以任何代价都无法通过暴力攻击破解的。[6]

注 5：1995 年之后，计算速度得到了飞速的提高，费用也降低了。你越晚读到这本书，计算的速度就会越快！但即使所需的时间会成 5 倍、10 倍或更多倍的减少，这张表仍然是有参考价值的。

注 6：但是，长的密钥并不意味着可以高枕无忧了！加密算法或实现中可能会有不为人注意的缺陷，为攻击者提供了可攻击的弱点。攻击者也可能会有一些与密钥产生方式有关的信息，这样他就会知道使用某些密钥的可能性比另一些要大，从而有助于进行有目的的暴力攻击。或者用户可能将保密的密钥落在了什么地方，被攻击者偷走了。

14.3.2 建立共享密钥

对称密钥加密技术的缺点之一就是发送者和接收者在互相对话之前，一定要有一个共享的保密密钥。

如果想要与 Joe 的五金商店进行保密的对话，可能是在看了公共电视台的家装节目之后，想要订购一些木工工具，那么在安全地订购任何东西之前，要先在你和 www.joes-hardware.com 之间建立一个私有的保密密钥。你需要一种产生保密密钥并将其记住的方式。你和 Joe 的五金商店，以及因特网上所有其他人，都要产生并记住数千个密钥。

比如 Alice（A）、Bob（B）和 Chris（C）都想与 Joe 的五金商店（J）对话。A、B 和 C 都要建立自己与 J 之间的保密密钥。A 可能需要密钥 K^{AJ}，B 可能需要密钥 K^{BJ}，C 可能需要密钥 K^{CJ}。每对通信实体都需要自己的私有密钥。如果有 N 个节点，每个节点都要和其他所有 N-1 个节点进行安全对话，总共大概会有 N^2 个保密密钥：这将是一个管理噩梦。

14.4 公开密钥加密技术

公开密钥加密技术没有为每对主机使用单独的加密 / 解密密钥，而是使用了两个非对称密钥：一个用来对主机报文编码，另一个用来对主机报文解码。编码密钥是众所周知的（这也是公开密钥加密这个名字的由来），但只有主机才知道私有的解密密钥（参见图 14-8）。这样，每个人都能找到某个特定主机的公开密钥，密钥的建立变得更加简单。但解码密钥是保密的，因此只有接收端才能对发送给它的报文进行解码。

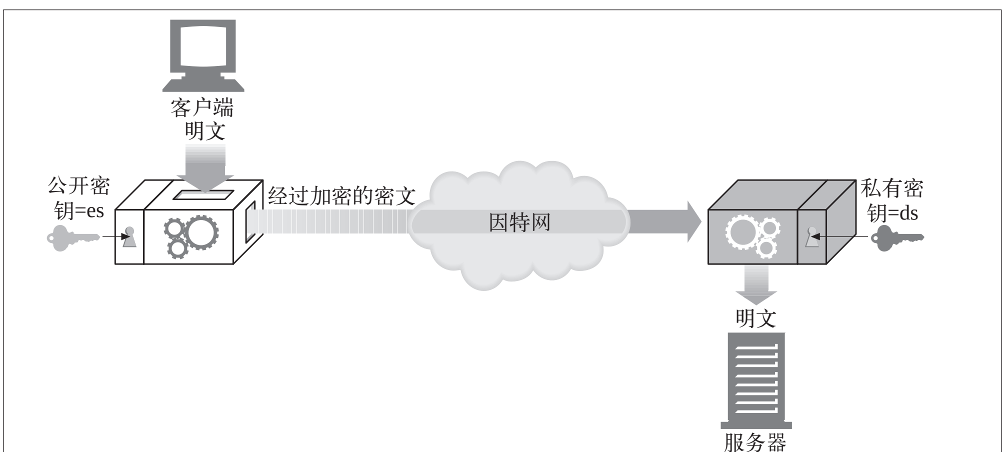


图 14-8　公开密钥加密技术是非对称的，为编码和解码使用了不同的密钥

节点 X 可以将其加密密钥 ex 公之于众。[7] 现在，任何想向节点 X 发送报文的人都可以使用相同的公开密钥了。因为每台主机都分配了一个所有人均可使用的编码密钥，所以公开密钥加密技术避免了对称密钥加密技术中成对密钥数目的 N^2 扩展问题（参见图 14-9）。 315

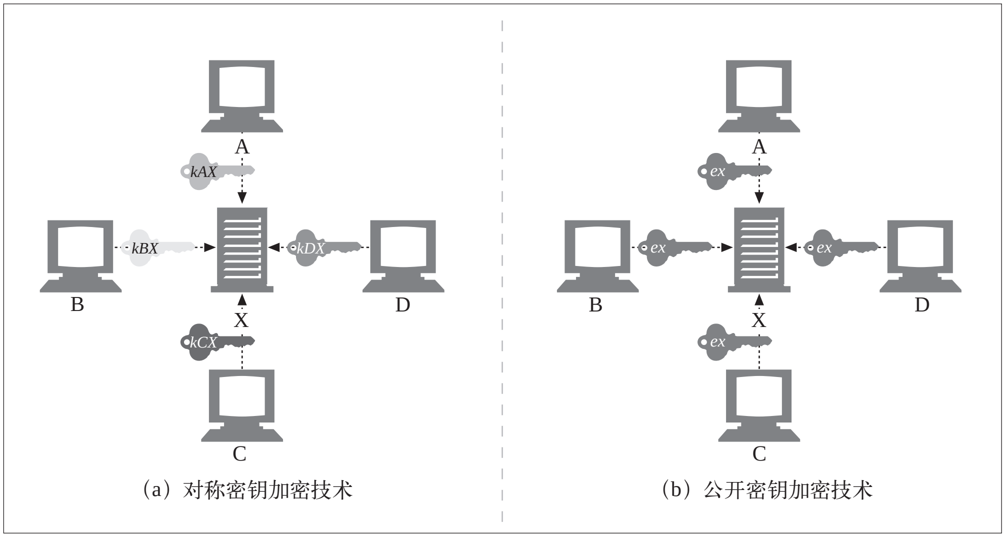


图 14-9　公开密钥加密技术为每台主机分配了一个公开编码密钥

尽管每个人都可以用同一个密钥对发给 X 的报文进行编码，但除了 X，其他人都无法对报文进行解码，因为只有 X 才有解码的私有密钥 d^x。将密钥分隔开来可以让所有人都能够对报文进行编码，但只有其所有者才能对报文进行解码。这样，各节点向服务器安全地发送报文就更加容易了，因为它们只要查找到服务器的公开密钥就行了。

通过公开密钥加密技术，全球所有的计算机用户就都可以使用安全协议了。制定标准化的公开密钥技术包是非常重要的，因此，大规模的公开密钥架构（Public-Key 316
Infrastructure，PKI）标准创建工作已经开展十多年了。

14.4.1　RSA

所有公开密钥非对称加密系统所面临的共同挑战是，要确保即便有人拥有了下面所有的线索，也无法计算出保密的私有密钥：

注 7：我们稍后会看到，大部分公开密钥查找工作实际上都是通过数字证书来实现的，但如何找到公开密钥现在并不重要——只要知道可以在某个地方公开获取就行了。

- 公开密钥（是公有的，所有人都可以获得）；
- 一小片拦截下来的密文（可通过对网络的嗅探获取）；
- 一条报文及与之相关的密文（对任意一段文本运行加密器就可以得到）。

RSA 算法就是一个满足了所有这些条件的流行的公开密钥加密系统，它是在 MIT 发明的，后来由 RSA 数据安全公司将其商业化。即使有了公共密钥、任意一段明文、用公共密钥对明文编码之后得到的相关密文、RSA 算法自身，甚至 RSA 实现的源代码，破解代码找到相应的私有密钥的难度仍相当于对一个极大的数进行质因数分解的困难程度，这种计算被认为是所有计算机科学中最难的问题之一。因此，如果你发现了一种能够快速地将一个极大的数字分解为质因数的方法，就不仅能够入侵瑞士银行的账户系统，而且还可以获得图灵奖了。

RSA 加密技术的细节中包括很多繁琐的数学问题，我们的介绍不会那么深入。你不需要拥有数论方面的博士学位，有大量的库可以用来执行 RSA 算法。

14.4.2　混合加密系统和会话密钥

任何人只要知道了其公开密钥，就可以向一台公共服务器发送安全报文，所以非对称的公开密钥加密系统是很好用的。两个节点无须为了进行安全的通信而先交换私有密钥。

但公开密钥加密算法的计算可能会很慢。实际上它混合使用了对称和非对称策略。比如，比较常见的做法是在两节点间通过便捷的公开密钥加密技术建立起安全通信，然后再用那条安全的通道产生并发送临时的随机对称密钥，通过更快的对称加密技术对其余的数据进行加密。

14.5　数字签名

到目前为止，我们已经讨论了各种使用对称和非对称密钥加 / 解密保密报文的密钥
317 加密技术。

除了加 / 解密报文之外，还可以用加密系统对报文进行*签名*（sign），以说明是谁编写的报文，同时证明报文未被篡改过。这种技术被称为*数字签名*（digital signing），对下一节将要讨论的因特网安全证书系统来说非常重要。

签名是加了密的校验和

数字签名是附加在报文上的特殊加密校验码。使用数字签名有以下两个好处。

- 签名可以证明是作者编写了这条报文。只有作者才会有最机密的私有密钥，[8] 因此，只有作者才能计算出这些校验和。校验和就像来自作者的个人“签名”一样。
- 签名可以防止报文被篡改。如果有恶意攻击者在报文传输过程中对其进行了修改，校验和就不再匹配了。由于校验和只有作者保密的私有密钥才能产生，所以攻击者无法为篡改了的报文伪造出正确的校验码。

数字签名通常是用非对称公开密钥技术产生的。因为只有所有者才知道其私有密钥，所以可以将作者的私有密钥当作一种“指纹”使用。

图 14-10 显示了一个例子，说明了节点 A 是如何向节点 B 发送一条报文，并对其进行签名的。

- 节点 A 将变长报文提取为定长的摘要。
- 节点 A 对摘要应用了一个“签名”函数，这个函数会将用户的私有密钥作为参数。因为只有用户才知道私有密钥，所以正确的签名函数会说明签名者就是其所有者。在图 14-10 中，由于解码函数 D 中包含了用户的私有密钥，所以我们将其作为签名函数使用。[9]
- 一旦计算出签名，节点 A 就将其附加在报文的末尾，并将报文和签名都发送给 B。

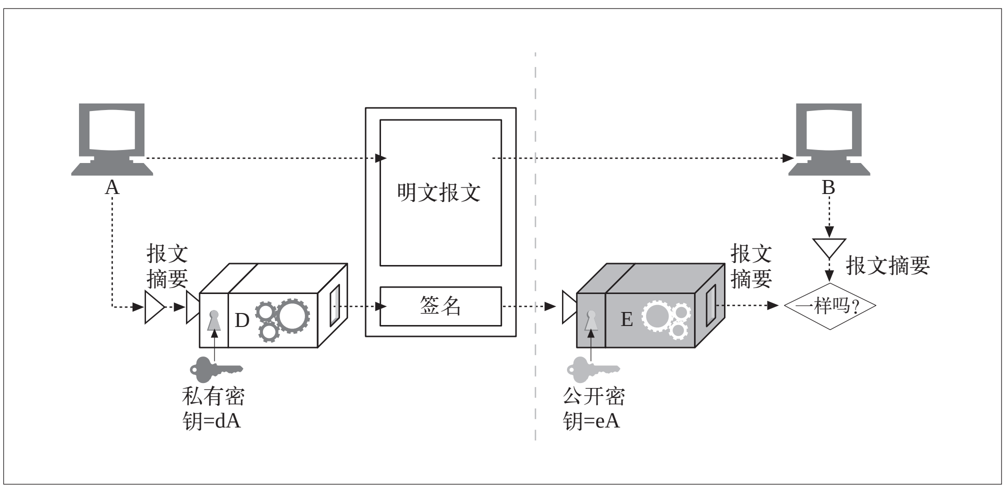


图 14-10　解密的数字签名

注 8：此时假定私有密钥没有被人偷走。大多数私有密钥都会在一段时间后过期。还有一些“取消列表”记录了被偷走或入侵的密钥。

注 9：RSA 加密系统将解码函数 D 作为签名函数使用，是因为 D 已经将私有密钥作为输入使用了。注意，解码函数只是一个函数，因此，可以将其用于任意的输入。同样，在 RSA 加密系统中，以任意顺序应用 D 和 E 函数时，两者都会相互抵消。因此 E(D(stuff)) = stuff，就像 D(E(stuff)) = stuff 一样。

- 在接收端，如果节点 B 需要确定报文确实是节点 A 写的，而且没有被篡改过，节点 B 就可以对签名进行检查。节点 B 接收经私有密钥扰码的签名，并应用了使用公开密钥的反函数。如果拆包后的摘要与节点 B 自己的摘要版本不匹配，要么就是报文在传输过程中被篡改了，要么就是发送端没有节点 A 的私有密钥（也
318 就是说它不是节点 A）。

14.6 数字证书

本节将介绍因特网上的“ID 卡”——数字证书。数字证书（通常被称作“certs”，有点像 certs 牌薄荷糖）中包含了由某个受信任组织担保的用户或公司的相关信息。

我们每个人都有很多形式的身份证明。有些 ID，比如护照和驾照，都足以在很多场合证明某人的身份。例如，你可以用美国的驾照在新年前夜搭乘前往纽约的航班，在你到那儿之后，接着用它来证明你的年龄，这样你就能和朋友们一起喝酒了。

受信程度更高的身份证明，比如护照，是由政府在特殊的纸上签发并盖章的。很难伪造，因此可以承载较高的信任度。有些公司的徽章和智能卡中包含有电子信息，以强化使用者的身份证明。有些绝密的政府组织甚至会对你的指纹或视网膜毛细血管模式进行匹配以便确认你的 ID！

有些形式的 ID，比如名片，相对来说更容易伪造，因此人们不太信任这些信息。虽然足以应付职场交流，但申请住房贷款时，可能就不足以证明你的就业情况了。

14.6.1 证书的主要内容

数字证书中还包含一组信息，所有这些信息都是由一个官方的“证书颁发机构”以数字方式签发的。基本的数字证书中通常包含一些纸质 ID 中常见的内容，比如：

- 对象的名称（人、服务器、组织等）；
- 319 过期时间；
- 证书发布者（由谁为证书担保）；
- 来自证书发布者的数字签名。

而且，数字证书通常还包括对象的公开密钥，以及对象和所用签名算法的描述性信息。任何人都可以创建一个数字证书，但并不是所有人都能够获得受人尊敬的签发权，从而为证书信息担保，并用其私有密钥签发证书。典型的证书结构如图 14-11 所示。

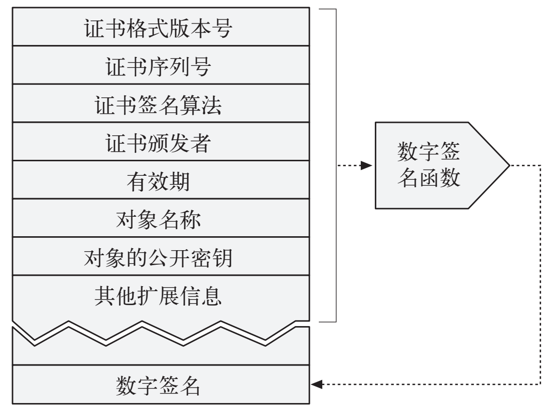


图 14-11　典型的数字签名格式

14.6.2　X.509 v3证书

不幸的是，数字证书没有单一的全球标准。就像不是所有印刷版 ID 卡都在同样的位置包含了同样的信息一样，数字证书也有很多略有不同的形式。 不过好消息就是现在使用的大多数证书都以一种标准格式——X.509 v3，来存储它们的信息。X.509 v3 证书提供了一种标准的方式，将证书信息规范至一些可解析字段中。不同类型的证书有不同的字段值，但大部分都遵循 X.509 v3 结构。表 14-2 介绍了 X.509 证书中的字段信息。

表14-2　X.509证书字段

字　段	描　述
版本	这个证书的 X.509 证书版本号。现在使用的通常都是版本 3
序列号	证书颁发机构（CA）生成的唯一整数。CA 生成的每个证书都要有一个唯一的序列号
签名算法 ID	签名所使用的加密算法。例如，“用 RSA 加密的 MD2 摘要”
证书颁发者	发布并签署这个证书的组织名称，以 X.500 格式表示
有效期	此证书何时有效，由一个起始日期和一个结束日期来表示
对象名称	证书中描述的实体，比如一个人或一个组织。对象名称是以 X.500 格式表示的
对象的公开密钥信息	证书对象的公开密钥，公开密钥使用的算法，以及所有附加参数
发布者唯一的 ID（可选）	可选的证书发布者唯一标识符，这样就可以重用相同的发布者名称
对象唯一的 ID（可选）	可选的证书对象唯一标识符，这样就可以重用相同的对象名称了

320

（续）

字　段	描　述
扩展	可选的扩展字段集（在版本 3 及更高的版本中使用）。每个扩展字段都被标识为关键或非关键的。关键扩展非常重要，证书使用者一定要能够理解。如果证书使用者无法识别出关键扩展字段，就必须拒绝这个证书。目前在使用的常用扩展字段包括： 基本约束 对象与证书颁发机构的关系 证书策略 授予证书的策略 密钥的使用 对公开密钥使用的限制
证书的颁发机构签名	证书颁发机构用指定的签名算法对上述所有字段进行的数字签名

基于 X.509 证书的签名有好几种，（其中）包括 Web 服务器证书、客户端电子邮件证书、软件代码签名证书和证书颁发机构证书。

14.6.3　用证书对服务器进行认证

通过 HTTPS 建立了一个安全 Web 事务之后，现代的浏览器都会自动获取所连接服务器的数字证书。如果服务器没有证书，安全连接就会失败。服务器证书中包含很多字段，其中包括：

- Web 站点的名称和主机名；
- Web 站点的公开密钥；
- 签名颁发机构的名称；
- 来自签名颁发机构的签名。

浏览器收到证书时会对签名颁发机构进行检查。[10] 如果这个机构是个很有权威的公
321 共签名机构，浏览器可能已经知道其公开密钥了（浏览器会预先安装很多签名颁发机构的证书）。这样，就可以像前面的 14.5 节中所讨论的那样验证签名了。图 14-12 说明了如何通过其数字签名来验证证书的完整性。

如果对签名颁发机构一无所知，浏览器就无法确定是否应该信任这个签名颁发机构，它通常会向用户显示一个对话框，看看他是否相信这个签名发布者。签名发布者可能是本地的 IT 部门或软件厂商。

注 10：浏览器和其他因特网应用程序都会尽量隐藏大部分证书管理的细节，使得浏览更加方便。但通过安全连接进行浏览时，所有主要的浏览器都允许你自己去检查所要对话站点的证书，以确保所有内容都是诚实可信的。

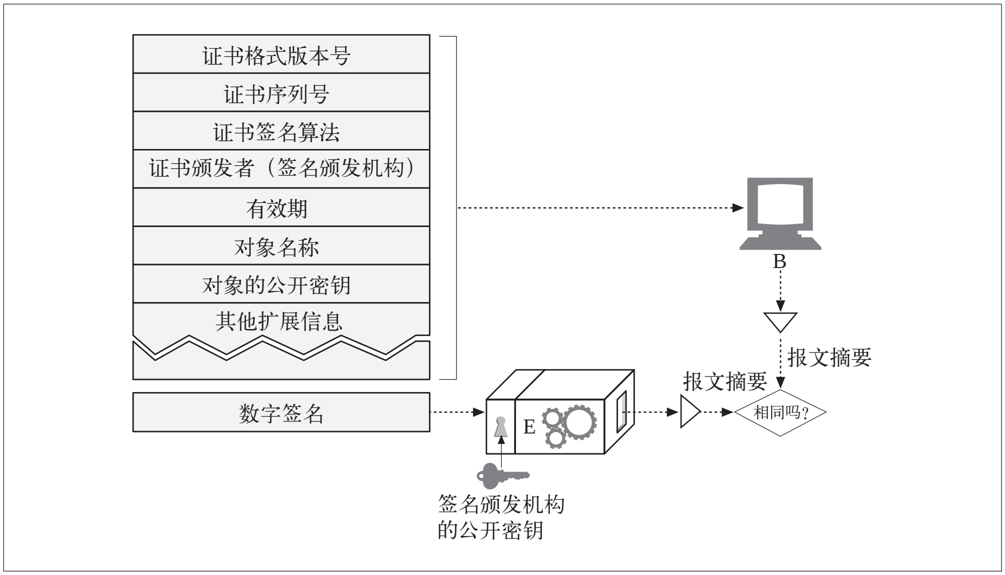


图 14-12　验证签名是真的

14.7　HTTPS——细节介绍

HTTPS 是最常见的 HTTP 安全版本。它得到了很广泛的应用，所有主要的商业浏览器和服务器上都提供 HTTPS。HTTPS 将 HTTP 协议与一组强大的对称、非对称和基于证书的加密技术结合在一起，使得 HTTPS 不仅很安全，而且很灵活，很容易在处于无序状态的、分散的全球互联网上进行管理。

HTTPS 加速了因特网应用程序的成长，已经成为基于 Web 的电子商务快速成长的主要推动力。在广域网中对分布式 Web 应用程序的安全管理方面，HTTPS 也是非常重要的。

14.7.1　HTTPS概述

HTTPS 就是在安全的传输层上发送的 HTTP。HTTPS 没有将未加密的 HTTP 报文发送给 TCP，并通过世界范围内的因特网进行传输（参见图 14-13a），它在将 HTTP 报文发送给 TCP 之前，先将其发送给了一个安全层，对其进行加密（参见
图 14-13b）。 322

现在，HTTP 安全层是通过 SSL 及其现代替代协议 TLS 来实现的。我们遵循常见的用法，用术语 SSL 来表示 SSL 或者 TLS。

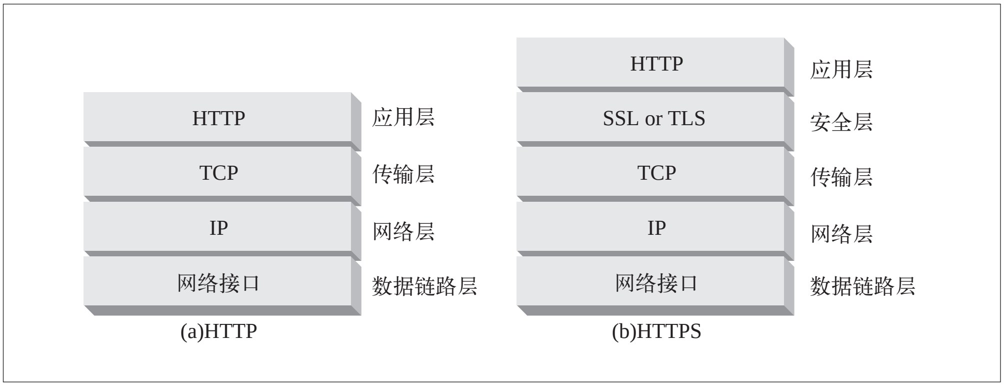


图 14-13　HTTP 传输层安全

14.7.2　HTTPS方案

现在，安全 HTTP 是可选的。因此，对 Web 服务器发起请求时，我们需要有一种方式来告知 Web 服务器去执行 HTTP 的安全协议版本。这是在 URL 的方案中实现的。

通常情况下，非安全 HTTP 的 URL 方案前缀为 http，如下所示：

http://www.joes-hardware.com/index.html

在安全 HTTPS 协议中，URL 的方案前缀为 https，如下所示：

https://cajun-shop.securesites.com/Merchant2/merchant.mv?Store_Code=AGCGS

请求一个客户端（比如 Web 浏览器）对某 Web 资源执行某事务时，它会去检查 URL 的方案。

- 如果 URL 的方案为 http，客户端就会打开一条到服务器端口 80（默认情况下）的连接，并向其发送老的 HTTP 命令（参见图 14-14a）。
- 如果 URL 的方案为 https，客户端就会打开一条到服务器端口 443（默认情况下）的连接，然后与服务器“握手”，以二进制格式与服务器交换一些 SSL 安全参数，附上加密的 HTTP 命令（参见图 14-14b）。

SSL 是个二进制协议，与 HTTP 完全不同，其流量是承载在另一个端口上的（SSL 通常是由端口 443 承载的）。如果 SSL 和 HTTP 流量都从端口 80 到达，大部分 Web 服务器会将二进制 SSL 流量理解为错误的 HTTP 并关闭连接。将安全服务进一步整合到 HTTP 层中去就无需使用多个目的端口了，在实际中这样不会引发严重的问题。

323 我们来详细介绍下 SSL 是如何与安全服务器建立连接的。

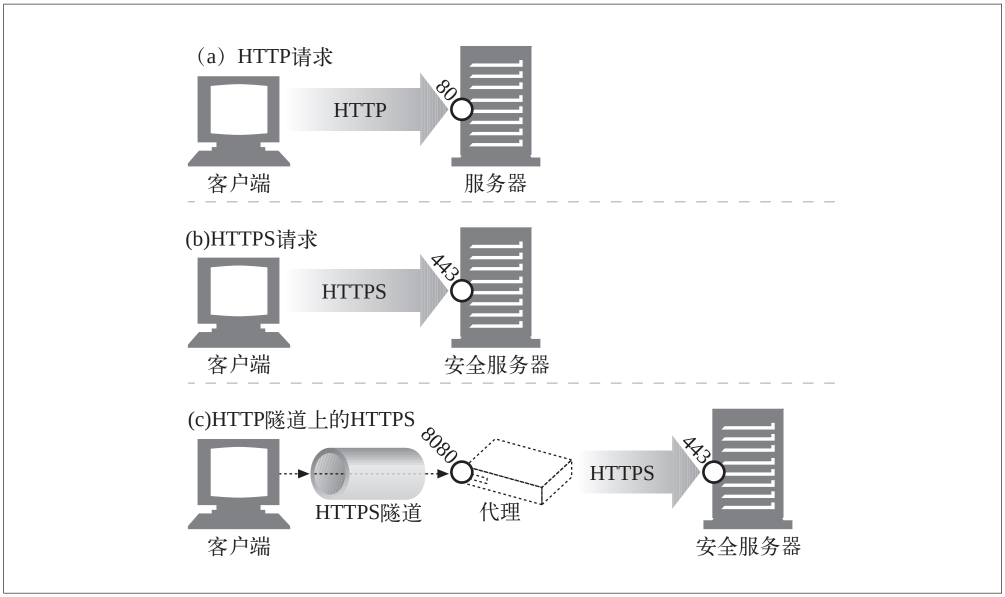


图 14-14　HTTP 和 HTTPS 端口号

14.7.3　建立安全传输

在未加密 HTTP 中，客户端会打开一条到 Web 服务器端口 80 的 TCP 连接，发送一条请求报文，接收一条响应报文，关闭连接。图 14-15a 对此序列进行了说明。

由于 SSL 安全层的存在，HTTPS 中这个过程会略微复杂一些。在 HTTPS 中，客户端首先打开一条到 Web 服务器端口 443（安全 HTTP 的默认端口）的连接。一旦建立了 TCP 连接，客户端和服务器就会初始化 SSL 层，对加密参数进行沟通，并交换密钥。握手完成之后，SSL 初始化就完成了，客户端就可以将请求报文发送给安全层了。在将这些报文发送给 TCP 之前，要先对其进行加密。图 14-15b 对此过程进行了说明。

14.7.4　SSL握手

在发送已加密的 HTTP 报文之前，客户端和服务器要进行一次 SSL 握手，在这个握手过程中，它们要完成以下工作：

- 交换协议版本号；
- 选择一个两端都了解的密码；
- 对两端的身份进行认证；
- 生成临时的会话密钥，以便加密信道。 324

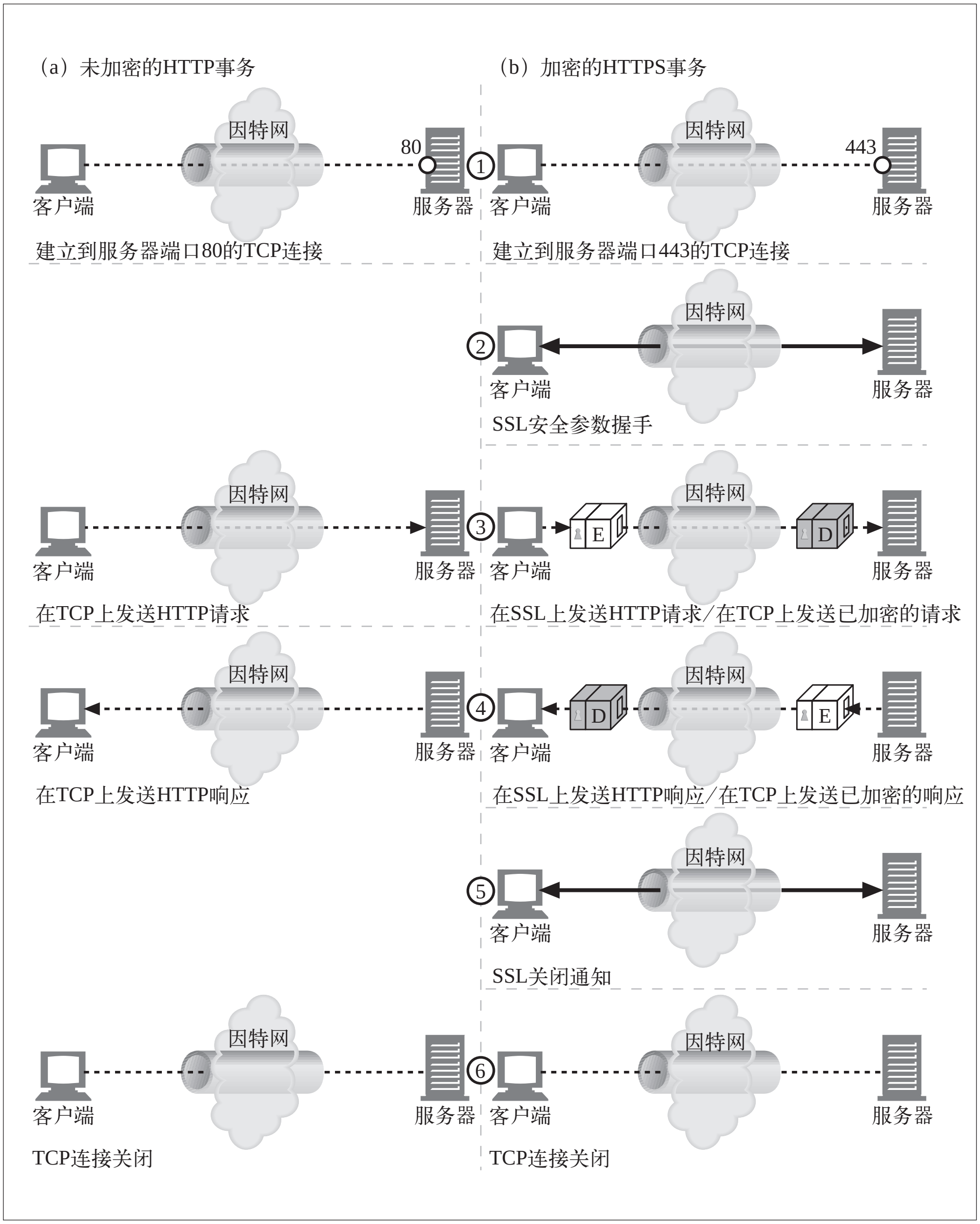


图 14-15　HTTP 和 HTTPS 事务

在通过网络传输任何已加密的 HTTP 数据之前，SSL 已经发送了一组握手数据来建立通信连接了。图 14-16 显示了 SSL 握手的基本思想。

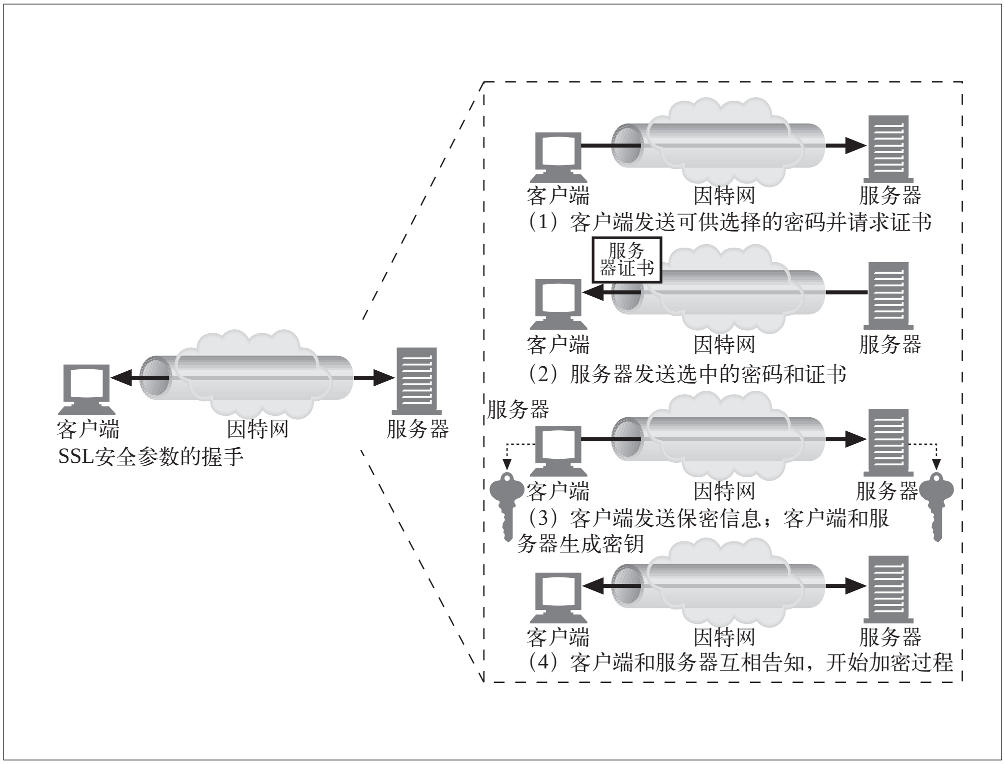


图 14-16 （简化版）SSL 握手

这是 SSL 握手的简化版本。根据 SSL 的使用方式，握手过程可能会复杂一些，但总的思想就是这样。 325

14.7.5 服务器证书

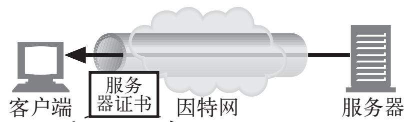
SSL 支持双向认证，将服务器证书承载回客户端，再将客户端的证书回送给服务器。而现在，浏览时并不经常使用客户端证书。大部分用户甚至都没有自己的客户端证书。[11] 服务器可以要求使用客户端证书，但实际中很少出现这种情况。[12]

另一方面，安全 HTTPS 事务总是要求使用服务器证书的。在一个 Web 服务器上执行安全事务，比如提交信用卡信息时，你总是希望是在与你所认为的那个组织对话。由知名权威机构签发的服务器证书可以帮助你在发送信用卡或私人信息之前评估你对服务器的信任度。

注 11：在某些公司的网络设置中会将客户端证书用于 Web 浏览，客户端证书还被用于安全电子邮件。未来，客户端证书可能会更经常地用于 Web 浏览，但现在它们发展的速度非常慢。

注 12：有些组织的内部网络会使用客户端证书来控制雇员对信息的访问。

服务器证书是一个显示了组织的名称、地址、服务器 DNS 域名以及其他信息的 X.509 v3 派生证书（参见图 14-17）。你和你所用的客户端软件可以检查证书，以确
326 保所有的信息都是可信的。



证书序列号	35:DE:F4:CF
证书过期时间	Wed,Sep 17,2003
站点的组织名	Joe’s Hardware Online
站点的DNS主机名	www.joes-hardware.com
站点的公开密钥	
证书颁发者的名称	RSA Data Security
证书颁发者的签名	John Doe

图 14-17　HTTPS 证书是带有站点信息的 X.509 证书

14.7.6　站点证书的有效性

SSL 自身不要求用户检查 Web 服务器证书，但大部分现代浏览器都会对证书进行简单的完整性检查，并为用户提供进行进一步彻查的手段。网景公司提出的一种 Web 服务器证书有效性算法是大部分浏览器有效性验证技术的基础。验证步骤如下所述。

- 日期检测

  首先，浏览器检查证书的起始日期和结束日期，以确保证书仍然有效。如果证书过期了，或者还未被激活，则证书有效性验证失败，浏览器显示一条错误信息。

- 签名颁发者可信度检测

  每个证书都是由某些证书颁发机构（CA）签发的，它们负责为服务器担保。证书有不同的等级，每种证书都要求不同级别的背景验证。比如，如果申请某个电子商务服务器证书，通常需要提供一个营业的合法证明。

  任何人都可以生成证书，但有些 CA 是非常著名的组织，它们通过非常清晰的流程来验证证书申请人的身份及商业行为的合法性。因此，浏览器会附带一个签名颁发机构的受信列表。如果浏览器收到了某未知（可能是恶意的）颁发机构

签发的证书，那它通常会显示一条警告信息。有些证书会携带到受信 CA 的有效签名路径，浏览器可能会选择接受所有此类证书。换句话说，如果某受信 CA 为“Sam 的签名商店”签发了一个证书，而 Sam 的签名商店也签发了一个站点证书，浏览器可能会将其作为从有效 CA 路径导出的证书接受。 327

- 签名检测

  一旦判定签名授权是可信的，浏览器就要对签名使用签名颁发机构的公开密钥，并将其与校验码进行比较，以查看证书的完整性。

- 站点身份检测

  为防止服务器复制其他人的证书，或拦截其他人的流量，大部分浏览器都会试着去验证证书中的域名与它们所对话的服务器的域名是否匹配。服务器证书中通常都包含一个域名，但有些 CA 会为一组或一群服务器创建一些包含了服务器名称列表或通配域名的证书。如果主机名与证书中的标识符不匹配，面向用户的客户端要么就去通知用户，要么就以表示证书不正确的差错报文来终止连接。

14.7.7 虚拟主机与证书

对虚拟主机（一台服务器上有多个主机名）站点上安全流量的处理有时是很棘手的。有些流行的 Web 服务器程序只支持一个证书。如果用户请求的是虚拟主机名，与证书名称并不严格匹配，浏览器就会显示警告框。

比如，我们来看以路易斯安那州为主题的电子商务网站 Cajun-Shop.com。站点的托管服务提供商提供的官方名称为 cajun-shop.securesites.com。用户进入 http://www.cajun-shop.com 时，服务器证书中列出的官方主机名（*.securesites.com）与用户浏览的虚拟主机名（www.cajun-shop.com）不匹配，以致出现图 14-18 中的警告。

为防止出现这个问题，Cajun-Shop.com 的所有者会在开始处理安全事务时，将所有用户都重定向到 cajun-shop.securesites.com。虚拟主机站点的证书管理会稍微棘手一些。

14.8 HTTPS客户端实例

SSL 是个复杂的二进制协议。除非你是密码专家，否则就不应该直接发送原始的 SSL 流量。幸运的是，借助一些商业或开源的库，编写 SSL 客户端和服务器并不十分困难。

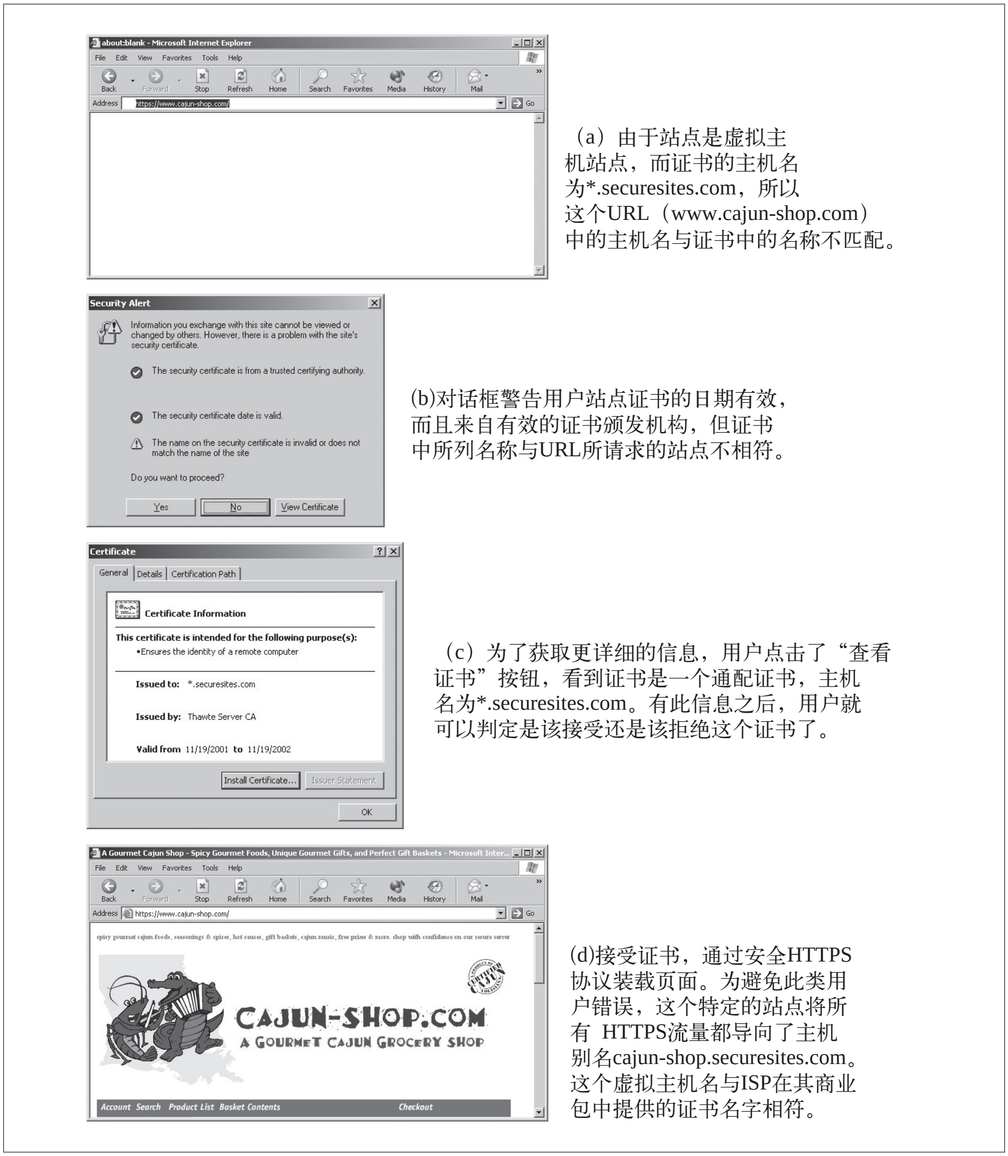


图 14-18　证书名不匹配引发的证书错误对话框

14.8.1　OpenSSL

OpenSSL 是 SSL 和 TLS 最常见的开源实现。OpenSSL 项目由一些志愿者合作开发，目标是开发一个强壮的、具有完备功能的商业级工具集，以实现 SSL 和 TLS 协议以及一个全功能的通用加密库。可以从 http://www.openssl.org 上获得 OpenSSL 的相
328 关信息，并下载相应软件。

你可能还听说过 SSLeay（读作 S-S-L-e-a-y）。OpenSSL 是 SSLeay 库的后继者，接口非常相似。SSLeay 最初是由 Eric A. Young（就是 SSLeay 中的“eay”）开发的。

14.8.2 简单的HTTPS客户端

本节我们将用 OpenSSL 包来编写一个非常初级的 HTTPS 客户端。这个客户端与服务器建立一条 SSL 连接，打印一些来自站点服务器的标识信息，通过安全信道发送 329 HTTP GET 请求，接收 HTTP 响应，并将响应打印出来。

下面显示的 C 程序是普通 HTTPS 客户端的 OpenSSL 实现。为了保持其简洁性，程序中没有包含差错处理和证书处理逻辑。

这个示例程序中删除了差错处理功能，所以只能将其用于示例。在一般的有差错存在的环境中，软件会崩溃或者无法正常运行。 330

```
/**********************************************************************
* https_client.c --- very simple HTTPS client with no error checking
*      usage: https_client servername
**********************************************************************/

#include <stdio.h>
#include <memory.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

void main(int argc, char **argv)
{

    SSL *ssl;
    SSL_CTX *ctx;
    SSL_METHOD *client_method;
    X509 *server_cert;
    int sd,err;
    char *str,*hostname,outbuf[4096],inbuf[4096],host_header[512];
    struct hostent *host_entry;
    struct sockaddr_in server_socket_address;
    struct in_addr ip;

    /*========================================*/
    /* (1) initialize SSL library */
    /*========================================*/
```

```
SSLeay_add_ssl_algorithms( );
client_method = SSLv2_client_method( );
SSL_load_error_strings( );
ctx = SSL_CTX_new(client_method);
printf("(1) SSL context initialized\n\n");
/*=============================================*/
/* (2) convert server hostname into IP address */
/*=============================================*/

hostname = argv[1];
host_entry = gethostbyname(hostname);
bcopy(host_entry->h_addr, &(ip.s_addr), host_entry->h_length);

printf("(2) '%s' has IP address '%s'\n\n", hostname, inet_ntoa(ip));
/*=================================================*/
/* (3) open a TCP connection to port 443 on server */
/*=================================================*/

sd = socket (AF_INET, SOCK_STREAM, 0);

memset(&server_socket_address, '\0', sizeof(server_socket_address));
server_socket_address.sin_family = AF_INET;
server_socket_address.sin_port = htons(443);
memcpy(&(server_socket_address.sin_addr.s_addr),
       host_entry->h_addr, host_entry->h_length);

err = connect(sd, (struct sockaddr*) &server_socket_address,
              sizeof(server_socket_address));
if (err < 0) { perror("can't connect to server port"); exit(1); }

printf("(3) TCP connection open to host '%s', port %d\n\n",
       hostname, server_socket_address.sin_port);

/*========================================================*/
/* (4) initiate the SSL handshake over the TCP connection */
/*========================================================*/

ssl = SSL_new(ctx);     /* create SSL stack endpoint */
SSL_set_fd(ssl, sd);    /* attach SSL stack to socket */
err = SSL_connect(ssl); /* initiate SSL handshake */

printf("(4) SSL endpoint created & handshake completed\n\n");

/*============================================*/
/* (5) print out the negotiated cipher chosen */
/*============================================*/

printf("(5) SSL connected with cipher: %s\n\n", SSL_get_cipher(ssl));

/*========================================*/
/* (6) print out the server's certificate */
/*========================================*/

server_cert = SSL_get_peer_certificate(ssl);
printf("(6) server's certificate was received:\n\n");
str = X509_NAME_oneline(X509_get_subject_name(server_cert), 0, 0);
printf("      subject: %s\n", str);
```

```
    str = X509_NAME_oneline(X509_get_issuer_name(server_cert), 0, 0);
    printf("      issuer: %s\n\n", str);

    /* certificate verification would happen here */

    X509_free(server_cert);

    /*********************************************************/
    /* (7) handshake complete --- send HTTP request over SSL */
    /*********************************************************/

    sprintf(host_header,"Host: %s:443\r\n",hostname);
    strcpy(outbuf,"GET / HTTP/1.0\r\n");
    strcat(outbuf,host_header);
    strcat(outbuf,"Connection: close\r\n");
    strcat(outbuf,"\r\n");

    err = SSL_write(ssl, outbuf, strlen(outbuf));
    shutdown (sd, 1); /* send EOF to server */

    printf("(7) sent HTTP request over encrypted channel:\n\n%s\n",outbuf);

    /**************************************************/
    /* (8) read back HTTP response from the SSL stack */
    /**************************************************/

    err = SSL_read(ssl, inbuf, sizeof(inbuf) - 1);
    inbuf[err] = '\0';
    printf ("(8) got back %d bytes of HTTP response:\n\n%s\n",err,inbuf);

    /************************************************/
    /* (9) all done, so close connection & clean up */
    /************************************************/

    SSL_shutdown(ssl);
    close (sd);
    SSL_free (ssl);
    SSL_CTX_free (ctx);

    printf("(9) all done, cleaned up and closed connection\n\n");
}
```

这个例子是在 Sun Solaris 上面编译运行的，但它说明了 SSL 在很多 OS 平台上的工作原理。由于 OpenSSL 提供了一些强有力的特性，包括所有加密、密钥以及证书管理在内的整个程序都可以在一个几页左右的 C 程序中实现。 331 ~ 332

下面按部分分析下这个程序。

- 程序的顶端包含了一些用于支持 TCP 联网和 SSL 的支撑文件。
- 第 1 部分通过调用 `SSL_CTX_new` 创建了本地上下文，以记录握手参数及与 SSL 连接有关的其他状态。
- 第 2 部分通过 Unix 的 `gethostbyname` 函数将（由一个命令行变元提供的）输入主机名转换成了 IP 地址。其他平台可能会通过其他方式来提供这项功能。

- 第 3 部分通过创建本地套接字、设置远端地址信息并连接到远端服务器，建立了一条到服务器端口 443 的 TCP 连接。
- 一旦 TCP 连接建立起来，就用 `SSL_new` 和 `SSL_set_fd` 将 SSL 层附加到 TCP 连接之上，并调用 `SSL_connect` 与服务器进行 SSL 握手。第 4 部分完成时，我们就建立了一个已选好密码且交换过证书的可运行的 SSL 信道。
- 第 5 部分打印了选中的批量加密密码值。
- 第 6 部分打印了服务器回送的 X.509 证书中包含的部分信息，其中包括与证书持有者和颁发证书的组织有关的信息。OpenSSL 库没有对服务器证书中的信息作任何特殊的处理。实际的 SSL 应用程序，比如 Web 浏览器会对证书进行一些完整性检查，以确保证书是正确签发的，且是来自正确主机的。我们在 14.7.6 节讨论了浏览器对服务器证书所做的处理。
- 此时，我们的 SSL 连接就已经可以用于安全数据的传输了。在第 7 部分中，用 `SSL_write` 在 SSL 信道上发送了简单的 HTTP 请求 GET / HTTP/1.0，然后关闭了连接的输出端。
- 在第 8 部分中，用 `SSL_read` 从连接上读回响应，并将其打印到屏幕上。SSL 层负责所有的加密和解密工作，因此可以直接读写普通的 HTTP 命令。
- 最后，在第 9 部分进行了一些清理工作。

更多与 OpenSSL 库有关的信息请参见 http://www.openssl.org。

14.8.3 执行OpenSSL客户端

下面显示了指向安全服务器时这个简单 HTTP 客户端的输出。在这个例子中，客户
333 端指向了摩根士丹利的在线证券主页。在线交易公司都在广泛使用 HTTPS。

```
% https_client clients1.online.msdw.com
(1) SSL context initialized

(2) 'clients1.online.msdw.com' has IP address'63.151.15.11'

(3) TCP connection open to host 'clients1.online.msdw.com', port 443

(4) SSL endpoint created & handshake completed

(5) SSL connected with cipher: DES-CBC3-MD5

(6) server's certificate was received:

      subject: /C=US/ST=Utah/L=Salt Lake City/O=Morgan Stanley/OU=Online/CN=
         clients1.online.msdw.com
      issuer: /C=US/O=RSA Data Security, Inc./OU=Secure Server Certification
         Authority
(7) sent HTTP request over encrypted channel:
```

```
GET / HTTP/1.0
Host: clients1.online.msdw.com:443
Connection: close

(8) got back 615 bytes of HTTP response:

HTTP/1.1 302 Found
Date: Sat, 09 Mar 2002 09:43:42 GMT
Server: Stronghold/3.0 Apache/1.3.14 RedHat/3013c (Unix) mod_ssl/2.7.1 OpenSSL/0.9.6
Location: https://clients.online.msdw.com/cgi-bin/ICenter/home
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved <A HREF="https://clients.online.msdw.com/cgi-bin/ICenter/
home">here</A>.<P>
<HR>
<ADDRESS>Stronghold/3.0Apache/1.3.14 RedHat/3013c Server at clients1.online.msdw.com
Port 443</ADDRESS>
</BODY></HTML>

(9) all done, cleaned up and closed connection
```

只要完成了前面 4 个部分，客户端就有了一条打开的 SSL 连接。这样它就可以查询连接的状态，选择参数，检查服务器证书了。

在这个例子中，客户端和服务器对 DES-CBC3-MD5 批量加密密码进行了沟通。你还能看到服务器站点证书属于美国犹他州盐湖城的摩根士丹利组织。证书由 RSA 数据安全组织授予，主机名为 clients1.online.msdw.com，与请求相符。 334

只要建立起了 SSL 信道，并且客户端对站点的证书没有异议，就可以通过安全信道来发送其 HTTP 请求了。在我们这个例子中，客户端发送了一条简单的“GET / HTTP/1.0”HTTP 请求，并收到了 302 Redirect 响应，请求用户去获取另一个 URL。

14.9 通过代理以隧道形式传输安全流量

客户端通常会用 Web 代理服务器（参见第 6 章）代表它们来访问 Web 服务器。比如，很多公司都会在公司网络和公共因特网的安全边界上放置一个代理（参见图 14-19）。代理是防火墙路由器唯一允许进行 HTTP 流量交换的设备，它可能会进行病毒检测或其他的内容控制工作。

但只要客户端开始用服务器的公开密钥对发往服务器的数据进行加密，代理就再也不能读取 HTTP 首部了！代理不能读取 HTTP 首部，就无法知道应该将请求转向何处了（参见图 14-20）。

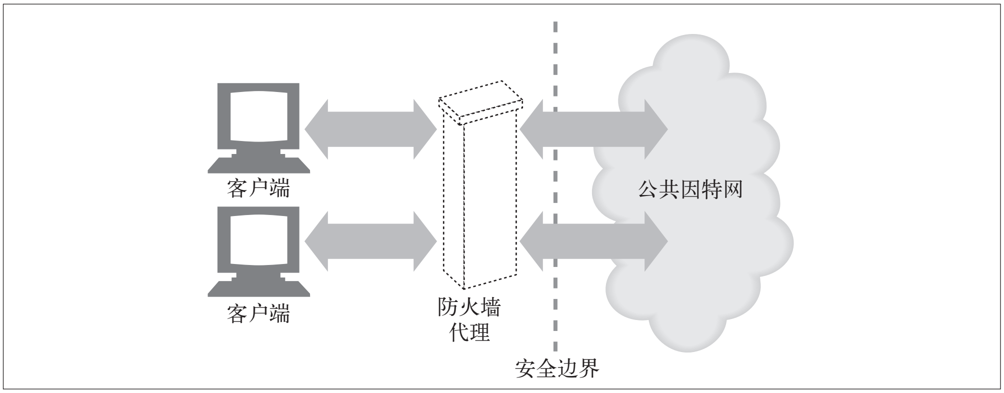


图 14-19　公司防火墙代理

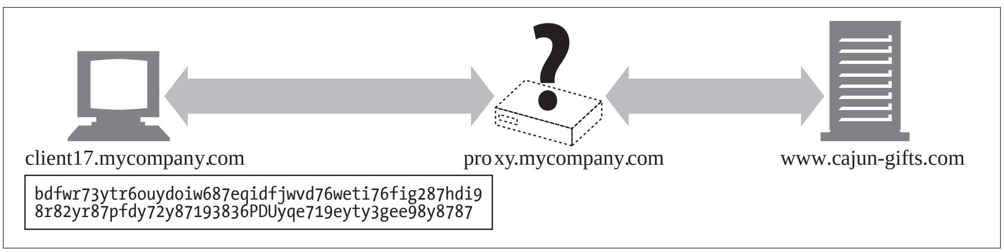


图 14-20　代理无法转发加密请求

为了使 HTTPS 与代理配合工作，要进行几处修改以告知代理连接到何处。一种常
335 用的技术就是 HTTPS SSL 隧道协议。使用 HTTPS 隧道协议，客户端首先要告知代理，它想要连接的安全主机和端口。这是在开始加密之前，以明文形式告知的，所以代理可以理解这条信息。

HTTP 通过新的名为 CONNECT 的扩展方法来发送明文形式的端点信息。CONNECT 方法会告诉代理，打开一条到所期望主机和端口号的连接。这项工作完成之后，直接在客户端和服务器之间以隧道形式传输数据。CONNECT 方法就是一条单行的文本命令，它提供了由冒号分隔的安全原始服务器的主机名和端口号。host:port 后面跟着一个空格和 HTTP 版本字符串，再后面是 CRLF。接下来是零个或多个 HTTP 请求首部行，后面跟着一个空行。空行之后，如果建立连接的握手过程成功完成，就可以开始传输 SSL 数据了。下面是一个例子：

```
CONNECT home.netscape.com:443 HTTP/1.0
User-agent: Mozilla/1.1N

<raw SSL-encrypted data would follow here...>
```

在请求中的空行之后，客户端会等待来自代理的响应。代理会对请求进行评估，确保它是有效的，而且用户有权请求这样一条连接。如果一切正常，代理会建立一条

到目标服务器的连接。如果成功，就向客户端发送一条 200 Connection Established 响应。

```
HTTP/1.0 200 Connection established
Proxy-agent: Netscape-Proxy/1.1
```

更多有关安全隧道和安全代理的信息，请回顾 8.5 节。

14.10　更多信息

安全和密码问题是非常重要，也非常复杂的问题。如果想学习更多有关 HTTP 安全性、数字加密技术、数字证书以及公开密钥基础设施方面的内容，可以从下面这几个地方开始。

14.10.1　HTTP安全性

- *Web security, Privacy & Commerce*[13]（《Web 安全与电子商务》）
  Simson Garfinkel 著，O'Reilly & Associates 公司。这是 Web 安全以及 SSL/TLS 和数字证书方面最好、最可读的入门型书籍之一。

- http://www.ietf.org/rfc/rfc2818.txt
  RFC 2818，“HTTP Over TLS”（“TLS 上的 HTTP”），说明了如何在 SSL 的后继协议——TLS 协议之上实现安全 HTTP。 336

- http://www.ietf.org/rfc/rfc2817.txt
  RFC 2817，“Upgrading to TLS Within HTTP/1.1”（“在 HTTP/1.1 中升级到 TLS”），说明了如何使用 HTTP/1.1 中的升级机制在现存的 TCP 连接上启动 TLS。这样非安全和安全 HTTP 流量就可以共享相同的知名端口了（在这种情况下，使用的是 http: 的 80 端口，而不是 https: 的 443 端口）。还可以使用虚拟主机技术。这样，使用一台 HTTP+TLS 服务器就可以区分出发往同一个 IP 地址上不同主机名的流量了。

14.10.2　SSL与TLS

- http://www.ietf.org/rfc/rfc2246.txt
  RFC 2246，“The TLS Protocol Version 1.0”（“TLS 协议版本 1.0”），对（SSL 的后继协议）TLS 协议的版本 1.0 进行了规范。TLS 提供了因特网上通信的私密性。协议允许客户端 / 服务器应用程序以防止窃听、篡改以及伪造报文的方式进行通信。

- http://developer.netscape.com/docs/manuals/security/sslin/contents.htm
  “Introduction to SSL”（“SSL 简介”）介绍了 SSL 协议。SSL 最初是由网景公司

注 13：本书中文版由中国电力出版社出版。（编者注）

开发的，已广泛应用于万维网上客户端和服务器间的认证及加密通信。

- http://www.netscape.com/eng/ssl3/draft302.txt
  “The SSL Protocol Version 3.0”（“SSL 协议版本 3.0”）是网景公司 1996 年的 SSL 规范。

- http://developer.netscape.com/tech/security/ssl/howitworks.html
  “How SSL Works”（“SSL 是如何工作的”）是网景公司对密钥加密技术的介绍。

- http://www.openssl.org
  OpenSSL 项目是一个合作开发项目，目的是开发一个强壮的、全功能的、商业级开源工具集，以实现安全套接字层（SSL v2/v3）和传输层安全（TLS v1）协议以及强大的通用密码库。这个项目由全世界范围内的志愿者社区管理，那些志愿者通过因特网进行交流、制定计划、开发 OpenSSL 工具集并撰写相关文档。OpenSSL 基于 Eric A. Young 和 Tim J. Hudson 开发的优秀 SSLeay 库。OpenSSL 工具集有一个 Apache 风格的许可证，这基本上就意味着只要遵循一些基本的许可条件，就可免费获得并将其用于商业或非商业目的。

14.10.3 公开密钥基础设施

- http://www.ietf.org/html.charters/pkix-charter.html
  IETF PKIX 工作组组建于 1995 年，目的是开发一些因特网标准，支持基于
  337 X.509 的公开密钥基础设施。这是对此小组活动很好的总结。

- http://www.ietf.org/rfc/rfc2459.txt
  RFC 2459，“Internet X.509 Public Key Infrastructure Certificate and CRL Profile”（“因特网 X.509 公开密钥基础设施证书及 CRL 概述”），详细介绍了 X.509 v3 数字证书。

14.10.4 数字密码

- *Applied Cryptography*[14]（《应用密码学》）
  Bruce Schneier 著，John Wiley & Sons 公司出版。这是为实现者编写的经典密码学书籍。

- *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*[15]（《密码故事——人类智力的另类较量》）
  Simon Singh 著，Anchor Books 公司出版。这是一本有趣的密码学入门书籍。它
  338 不是为技术专家编写的，而是一本生动的密码学历史读物。

注 14：本书中文版由机械工业出版社出版。（编者注）
注 15：本书中文版由海南出版社出版。（编者注）

第四部分

实体、编码和国际化

第四部分讲述的内容都与 HTTP 报文的实体主体和被实体主体作为货物承载的内容有关。

- 第 15 章讲述了 HTTP 内容的格式和语法。
- 第 16 章探讨了允许世界各地的人们相互交换内容的各种 Web 标准，这些内容由各种不同语言和不同字符集构成。 339～340
- 第 17 章讲解了各种用于协商可接受内容的机制。

第15章

实体和编码

每天都有数以亿计的各种媒体对象经由 HTTP 传送，如图像、文本、影片以及软件程序等。只要你能叫出名字，HTTP 就可以传送。HTTP 还会确保它的报文被正确传送、识别、提取以及适当处理。具体说来，HTTP 要确保它所承载的“货物”满足以下条件。

- 可以被正确地识别（通过 Content-Type 首部说明媒体格式，Content-Language 首部说明语言），以便浏览器和其他客户端能正确处理内容。
- 可以被正确地解包（通过 Content-Length 首部和 Content-Encoding 首部）。
- 是最新的（通过实体验证码和缓存过期控制）
- 符合用户的需要（基于 Accept 系列的内容协商首部）
- 在网络上可以快速有效地传输（通过范围请求、差异编码以及其他数据压缩方法）
- 完整到达、未被篡改（通过传输编码首部和 Content-MD5 校验和首部）

为了实现这些目标，HTTP 使用了完善的标签来描述承载内容的实体。

本章讨论各种实体、与它们相关的实体首部以及它们是如何运作来传送网站“货物”的。我们将展示 HTTP 是如何提供必需的内容大小、类型以及编码的。我们还要解释 HTTP 实体的某些更复杂和强大的特性，比如范围请求、差异编码、摘要以及分块编码等。

本章涵盖了下列内容。

- 作为 HTTP 数据的容器，HTTP 报文实体有哪些格式和行为。
- HTTP 如何描述实体的主体大小，HTTP 为确定大小制定了哪些规则。
- 为了使客户端正确处理内容，使用了哪些实体首部来描述内容的格式、字母和语言。
341
- 可逆的内容编码，发送方可以在发送之前用它来转换内容的数据格式，使其占用更小的空间，或者更安全。
- 传输编码和分块编码。传输编码可以改变 HTTP 传输数据的方式，以改善某些类型内容的通信能力。分块编码是一种特殊的传输编码，它把数据切分为若干块，这样可以更可靠地传输长度未知的内容。
- 标记、标签、时间以及校验和等一整套机制，帮助客户端获取所请求内容的最新版本。
- 可用作内容版本号的验证码，网站应用可以通过它确保接收最新的内容。还有设计用来控制对象新鲜度的各种 HTTP 首部字段。
- 范围，在恢复中断的传输方面很有用。
- HTTP 差异编码扩展，它使客户端只需要请求网页中和前一次相比有改变的部分。
- 实体主体的校验和，可以用来检测经过若干代理之后，实体的内容是否发生了改变。

15.1　报文是箱子，实体是货物

如果把 HTTP 报文想象成因特网货运系统中的箱子，那么 HTTP 实体就是报文中实际的货物。图 15-1 展示了一个简单的实体，装在 HTTP 响应报文中。

```
HTTP/1.0 200 OK
Server: Netscape-Enterprise/3.6
Date: Sun, 17 Sep 2000 00:01:05 GMT
Content-type: text/plain          实体首部
Content-length: 18
                                            实体
Hi! I'm a message!                实体主体
```

图 15-1　报文实体由实体首部和实体主体组成

实体首部指出这是一个纯文本文档（`Content-Type:text/plain`），它只有 18 个字节长（`Content-Length:18`）。和往常一样，一个空白行（CRLF）把首部字段同主体的开始部分分隔开来。

HTTP 实体首部（参见第 3 章）描述了 HTTP 报文的内容。HTTP/1.1 版定义了以下 10 个基本字体首部字段。

- `Content-Type`
  实体中所承载对象的类型。
- `Content-Length`
  所传送实体主体的长度或大小。 342
- `Content-Language`
  与所传送对象最相配的人类语言。
- `Content-Encoding`
  对象数据所做的任意变换（比如，压缩）。
- `Content-Location`
  一个备用位置，请求时可通过它获得对象。
- `Content-Range`
  如果这是部分实体，这个首部说明它是整体的哪个部分。
- `Content-MD5`
  实体主体内容的校验和。

- Last-Modified
  所传输内容在服务器上创建或最后修改的日期时间。

- Expires
  实体数据将要失效的日期时间。

- Allow
  该资源所允许的各种请求方法，例如，GET 和 HEAD。

- ETag
  这份文档特定实例（参见 15.7 节）的唯一验证码。ETag 首部没有正式定义为实体首部，但它对许多涉及实体的操作来说，都是一个重要的首部。

- Cache-Control
  指出应该如何缓存该文档。和 ETag 首部类似，Cache-Control 首部也没有正式定义为实体首部。

实体主体

实体主体中就是原始货物啦。[1] 任何其他描述性的信息都包含在首部中。因为货物（也就是实体主体）只是原始数据，所以需要实体首部来描述数据的意义。例如，Content-Type 实体首部告诉我们如何去解释数据（是图像还是文本等），而 Content-Encoding 实体首部告诉我们数据是不是已被压缩或者重编码。我们将在随后的小节中讨论所有这些方面及更多的内容。

首部字段以一个空白的 CRLF 行结束，随后就是实体主体的原始内容。不管内容是什么，文本或二进制的、文档或图像、压缩的或未压缩的、英语、法语或日语，都
343 紧随这个 CRLF 之后。

图 15-2 展示了两个实际的 HTTP 报文的例子。一个携带着文本实体，另一个承载的是图像实体。十六进制的数值中展示的是报文的实际内容。

- 在图 15-2a 中，实体主体从第 65 个字节开始，紧随首部末尾的 CRLF。实体主体中包含了“Hi! I'm a message!”这句话的 ASCII 编码字符。
- 在图 15-2b 中，实体主体从第 67 字节开始。实体主体包含了一个 GIF 格式图像的二进制内容。GIF 文件以 6 个字节的版本标志开头，后面是 16 位的宽度和 16 位的高度。可以在实体主体中直接看到这 3 项内容。

注 1：如果有 Content-Encoding 首部的话，实体主体的内容就已经被指定的内容编码算法进行过编码了，第一个字节就是编码（比如，压缩）后的货物的第一个字节。



图 15-2　实际报文内容（紧随空白的 CRLF 之后的原始报文内容）的十六进制转储显示

15.2　`Content-Length`：实体的大小

`Content-Length` 首部指示出报文中实体主体的字节大小。这个大小是包含了所有内容编码的，比如，对文本文件进行了 gzip 压缩的话，`Content-Length` 首部就是压缩后的大小，而不是原始大小。

除非使用了分块编码，否则 `Content-Length` 首部就是带有实体主体的报文必须使用的。使用 `Content-Length` 首部是为了能够检测出服务器崩溃而导致的报文截尾，并对共享持久连接的多个报文进行正确分段。

15.2.1　检测截尾

HTTP 的早期版本采用关闭连接的办法来划定报文的结束。但是，没有 `Content-Length` 的话，客户端无法区分到底是报文结束时正常的连接关闭，还是报文传输中由于服务器崩溃而导致的连接关闭。客户端需要通过 `Content-Length` 来检测报文截尾。 344

报文截尾的问题对缓存代理服务器来说尤其严重。如果缓存服务器收到被截尾的报文却没有识别出截尾的话，它可能会存储不完整的内容并多次使用它来提供服务。缓存代理服务器通常不会为没有显式 `Content-Length` 首部的 HTTP 主体做缓存，以此来减小缓存已截尾报文的风险。

15.2.2 错误的Content-Length

错误的 Content-Length 比缺少 Content-Length 还要糟糕。因为某些早期的客户端和服务器在 Content-Length 计算上存在一些众所周知的错误，有些客户端、服务器以及代理中就包含了特别的算法，用来检测和纠正与有缺陷服务器的交互过程。HTTP/1.1 规定用户 Agent 代理应该在接收且检测到无效长度时通知用户。

15.2.3 Content-Length与持久连接

Content-Length 首部对于持久连接是必不可少的。如果响应通过持久连接传送，就可能有另一条 HTTP 响应紧随其后。客户端通过 Content-Length 首部就可以知道报文在何处结束，下一条报文从何处开始。因为连接是持久的，客户端无法依赖连接关闭来判别报文的结束。如果没有 Content-Length 首部，HTTP 应用程序就不知道某个实体主体在哪里结束，下一条报文从哪里开始。

我们将在 15.6 节看到，有一种情况下，使用持久连接时可以没有 Content-Length 首部，即采用*分块编码*（chunked encoding）时。在分块编码的情况下，数据是分为一系列的块来发送的，每块都有大小说明。哪怕服务器在生成首部的时候不知道整个实体的大小（通常是因为实体是动态生成的），仍然可以使用分块编码传输若干已知大小的块。

15.2.4 内容编码

HTTP 允许对实体主体的内容进行编码，比如可以使之更安全或进行压缩以节省空间（本章稍后将详细解释压缩的问题）。如果主体进行了内容编码，Content-Length 首部说明的就是*编码后*（encoded）的主体的字节长度，而*不是*未编码的原始主体的长度。

某些 HTTP 应用程序在这方面搞错了，发送的是数据编码之前的大小，这会导致严
345 重的错误，尤其是用在持久连接上。不幸的是，HTTP/1.1 规范中没有首部可以用来说明原始的、未编码的主体的长度，这就让客户端难以验证*解码过程*的完整性。[2]

15.2.5 确定实体主体长度的规则

下面列出的规则说明了在若干不同的情况下如何正确计算主体的长度和结束位置。这些规则应当按顺序应用，谁先匹配就用谁。

注 2：Content-MD5 首部也不行，它用来说明文档的 128 位 MD5 值，但这也是针对编码后的文档的。本章后面会说明 Content-MD5 首部。

（1）如果特定的 HTTP 报文类型中不允许带有主体，就忽略 `Content-Length` 首部，它是对（没有实际发送出来的）主体进行计算的。这种情况下，`Content-Length` 首部是提示性的，并不说明实际的主体长度。（考虑不周的 HTTP 应用程序会认为有了 `Content-Length` 就有主体存在，这样就会出问题。）

最重要的例子就是 HEAD 响应。HEAD 方法请求服务器发送等价的 GET 请求中会出现的首部，但不要包括主体。因为对 GET 的响应会带有 `Content-Length` 首部，所以 HEAD 响应里面也有；但和 GET 响应不同的是，HEAD 响应中不会有主体。1XX、204 以及 304 响应也可以有提示性的 `Content-Length` 首部，但是也都没有实体主体。那些规定不能带有实体主体的报文，不管带有什么首部字段，都必须在首部之后的第一个空行终止。

（2）如果报文中含有描述传输编码的 `Transfer-Encoding` 首部（不采用默认的 HTTP“恒等”编码），那实体就应由一个称为“零字节块”（zero-byte chunk）的特殊模式结束，除非报文已经因连接关闭而结束。我们将在本章后面讨论传输编码和分块编码。

（3）如果报文中含有 `Content-Length` 首部（并且报文类型允许有实体主体），而且没有非恒等的 `Transfer-Encoding` 首部字段，那么 `Content-Length` 的值就是主体的长度。如果收到的报文中既有 `Content-Length` 首部字段又有非恒等的 `Transfer-Encoding` 首部字段，那就必须忽略 `Content-Length`，因为传输编码会改变实体主体的表示和传输方式（因此可能就会改变传输的字节数）。

（4）如果报文使用了 multipart/byteranges（多部分 / 字节范围）媒体类型，并且没有用 `Content-Length` 首部指出实体主体的长度，那么多部分报文中的每个部分都要说明它自己的大小。这种多部分类型是唯一的一种自定界的实体主体类型，因此除非发送方知道接收方可以解析它，否则就不能发送这种媒体类型。[3] 346

（5）如果上面的规则都不匹配，实体就在连接关闭的时候结束。实际上，只有服务器可以使用连接关闭来指示报文的结束。客户端不能用关闭连接来指示客户端报文的结束，因为这样会使服务器无法发回响应。[4]

为了和使用 HTTP/1.0 的应用程序兼容，任何带有实体主体的 HTTP/1.1 请求都必须带有正确的 `Content-Length` 首部字段（除非已经知道服务器兼容 HTTP/1.1）。

注 3：因为 `Range` 首部可能会被不理解多部分 / 字节范围的更原始的代理所转发，所以如果发送方不能确定接收方是否理解这种自定界的格式的话，就必须用本节的方法（1）、（3）或（5）来对报文定界。

注 4：客户端可以使用半关闭，也就是只把连接的输出端关闭，但很多服务器应用程序设计的时候没有考虑到处理这种情况，会把半关闭当作客户端要从服务器断开连接来处理。HTTP 没有对连接管理进行良好的规范。详情请参见第 4 章。

HTTP/1.1 规范中建议对于带有主体但没有 `Content-Length` 首部的请求，服务器如果无法确定报文的长度，就应当发送 400 Bad Request 响应或 411 Length Required 响应，后一种情况表明服务器要求收到正确的 `Content-Length` 首部。

15.3 实体摘要

尽管 HTTP 通常都是在像 TCP/IP 这样的可靠传输协议之上实现的，但仍有很多因素会导致报文的一部分在传输过程中被修改，比如有不兼容的转码代理，或者中间代理有误，等等。为检测实体主体的数据是否被不经意（或不希望有）地修改，发送方可以在生成初始的主体时，生成一个数据的校验和，这样接收方就可以通过检查这个校验和来捕获所有意外的实体修改了。[5]

服务器使用 `Content-MD5` 首部发送对实体主体运行 MD5 算法的结果。只有产生响应的原始服务器可以计算并发送 `Content-MD5` 首部。中间代理和缓存不应当修改或添加这个首部，否则就会与验证端到端完整性的这个最终目的相冲突。`Content-MD5` 首部是在对内容做了所有需要的内容编码之后，还没有做任何传输编码之前，计算出来的。为了验证报文的完整性，客户端必须先进行传输编码的解码，然后计算所得到的未进行传输编码的实体主体的 MD5。举个例子吧，如果一份文档使用 gzip 算法进
347 行压缩，然后用分块编码发送，那么就对整个经 gzip 压缩的主体进行 MD5 计算。

除了检查报文的完整性之外，MD5 还可以当作散列表的关键字，用来快速定位文档并消除不必要的重复内容存储。除了这些可能的用法，一般不常用到 `Content-MD5` 首部。

作为对 HTTP 的扩展，在 IETF 的草案中提出了其他一些摘要算法。这些扩展建议增加新的 `Want-Digest` 首部，它允许客户端说明期望响应中使用的摘要类型，并使用质量值来建议多种摘要算法并说明优先顺序。

15.4 媒体类型和字符集

`Content-Type` 首部字段说明了实体主体的 MIME 类型。[6]MIME 类型是标准化的名字，用以说明作为货物运载实体的基本媒体类型（比如：HTML 文件、Microsoft Word 文档或是 MPEG 视频等）。客户端应用程序使用 MIME 类型来解释和处理其内容。

注 5：当然，这种方法对同时替换报文主体和摘要首部的恶意攻击无效。这只是为了检测不经意的修改。对付恶意篡改，需要使用别的机制，比如摘要认证。

注 6：在 HEAD 请求中，`Content-Type` 说明如果请求是 GET 时，将要发送的主体的类型。

Content-Type 的值是标准化的 MIME 类型，都在互联网号码分配机构（Internet Assigned Numbers Authority，简称 IANA）中注册。MIME 类型由一个主媒体类型（比如：text、image 或 audio 等）后面跟一条斜线以及一个子类型组成，子类型用于进一步描述媒体类型。表 15-1 中列出了一些 Content-Type 首部中常用的 MIME 类型。附录 D 中列出了更多的 MIME 类型。

表15-1　常用媒体类型

媒体类型	描　述
text/html	实体主体是 HTML 文档
text/plain	实体主体是纯文本文档
image/gif	实体主体是 GIF 格式的图像
image/jpeg	实体主体是 JPEG 格式的图像
audio/x-wav	实体主体包含 WAV 格式声音数据
model/vrml	实体主体是三维的 VRML 模型
application/vnd.ms-powerpoint	实体主体是 Microsoft PowerPoint 演示文档
multipart/byteranges	实体主体有若干部分，每个部分都包含了完整文档中不同的字节范围
message/http	实体主体包含完整的 HTTP 报文（参见 TRACE）

要着重注意的是，Content-Type 首部说明的是原始实体主体的媒体类型。例如，如果实体经过内容编码的话，Content-Type 首部说明的仍是编码之前的实体主体的类型。 348

15.4.1　文本的字符编码

Content-Type 首部还支持可选的参数来进一步说明内容的类型。charset（字符集）参数就是个例子，它说明把实体中的比特转换为文本文件中的字符的方法：

```
Content-Type: text/html; charset=iso-8859-4
```

我们将在第 16 章详细讨论字符集。

15.4.2　多部分媒体类型

MIME 中的 multipart（多部分）电子邮件报文中包含多个报文，它们合在一起作为单一的复杂报文发送。每一部分都是独立的，有各自的描述其内容的集；不同的部分之间用分界字符串连接在一起。

HTTP 也支持多部分主体。不过，通常只用在下列两种情形之一：提交填写好的表格，或是作为承载若干文档片段的范围响应。

15.4.3 多部分表格提交

当提交填写的 HTTP 表格时，变长的文本字段和上传的对象都作为多部分主体里面独立的部分发送，这样表格中就可以填写各种不同类型和长度的值。比如，你可能选择用昵称和小照片来填写询问你的名字和介绍信息的表格，而你的朋友可能填了她的全名并在介绍信息表内抱怨了一堆大众汽车的修理问题。

HTTP 使用 `Content-Type:multipart/form-data` 或 `Content-Type:multipart/mixed` 这样的首部以及多部分主体来发送这种请求，举例如下：

```
Content-Type: multipart/form-data; boundary=[abcdefghijklmnopqrstuvwxyz]
```

其中的 `boundary` 参数说明了分割主体中不同部分所用的字符串。

349 下面的例子展示了 `multipart/form-data` 编码。假设我们有这样的表格：

```
<FORM action=" http://server.com/cgi/handle"
      enctype="multipart/form-data"
      method="post">
<P>
What is your name? <INPUT type="text" name="submit-name"><BR>
What files are you sending? <INPUT type="file" name="files"><BR>
<INPUT type="submit" value="Send"> <INPUT type="reset">
</FORM>
```

如果用户在文本输入字段中键入 Sally，并选择了文本文件 essayfile.txt，用户 Agent 代理可能会发回下面这样的数据：

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"
Sally
--AaB03x
Content-Disposition: form-data; name="files"; filename="essayfile.txt"
Content-Type: text/plain
...contents of essayfile.txt...
--AaB03x--
```

如果用户还选了另一个（图像）文件 imagefile.gif，用户 Agent 代理可能像下面这样构造这个部分：

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"
Sally
--AaB03x
Content-Disposition: form-data; name="files"
Content-Type: multipart/mixed; boundary=BbC04y
```

```
--BbC04y
Content-Disposition: file; filename="essayfile.txt"
Content-Type: text/plain
...contents of essayfile.txt...
--BbC04y
Content-Disposition: file; filename="imagefile.gif"
Content-Type: image/gif
Content-Transfer-Encoding: binary
...contents of imagefile.gif...
--BbC04y--
--AaB03x--
```

15.4.4 多部分范围响应

HTTP 对范围请求的响应也可以是多部分的。这样的响应中有 Content-Type: multipart/byteranges 首部和带有不同范围的多部分主体。下面是一个例子，展示了对文档不同范围的请求产生的响应：

```
HTTP/1.0 206 Partial content
Server: Microsoft-IIS/5.0
Date: Sun, 10 Dec 2000 19:11:20 GMT
Content-Location: http://www.joes-hardware.com/gettysburg.txt
Content-Type: multipart/x-byteranges; boundary=--[abcdefghijklmnopqrstu
vwxyz]--
Last-Modified: Sat, 09 Dec 2000 00:38:47 GMT

--[abcdefghijklmnopqrstuvwxyz]--
Content-Type: text/plain
Content-Range: bytes 0-174/1441
Fourscore and seven years ago our fathers brought forth on this
continent a new nation, conceived in liberty and dedicated to the
proposition that all men are created equal.
--[abcdefghijklmnopqrstuvwxyz]--
Content-Type: text/plain
Content-Range: bytes 552-761/1441

But in a larger sense, we can not dedicate, we can not consecrate,
we can not hallow this ground. The brave men, living and dead who
struggled here have consecrated it far above our poor power to add
or detract.
--[abcdefghijklmnopqrstuvwxyz]--
Content-Type: text/plain
Content-Range: bytes 1344-1441/1441

and that government of the people, by the people, for the people shall
not perish from the earth.

--[abcdefghijklmnopqrstuvwxyz]--
```

350

本章后面将详细讨论范围请求。

15.5 内容编码

HTTP 应用程序有时在发送之前需要对内容进行编码。例如，在把很大的 HTML 文档发送给通过慢速连接连上来的客户端之前，服务器可能会对它进行压缩，这样有助于减少传输实体的时间。服务器还可以把内容搅乱或加密，以此来防止未经授权的第三方看到文档的内容。

这种类型的编码是在发送方应用到内容之上的。当内容经过内容编码之后，编好码的数据就放在实体主体中，像往常一样发送给接收方。

15.5.1 内容编码过程

内容编码的过程如下所述。

（1）网站服务器生成原始响应报文，其中有原始的 `Content-Type` 和 `Content-Length` 首部。

（2）内容编码服务器（也可能就是原始的服务器或下行的代理）创建编码后的报文。编码后的报文有同样的 `Content-Type` 但 `Content-Length` 可能不同（比如主体被压缩了）。内容编码服务器在编码后的报文中增加 `Content-Encoding` 首部，这样接收的应用程序就可以进行解码了。

351 （3）接收程序得到编码后的报文，进行解码，获得原始报文。

图 15-3 给出了内容编码的梗概示例。

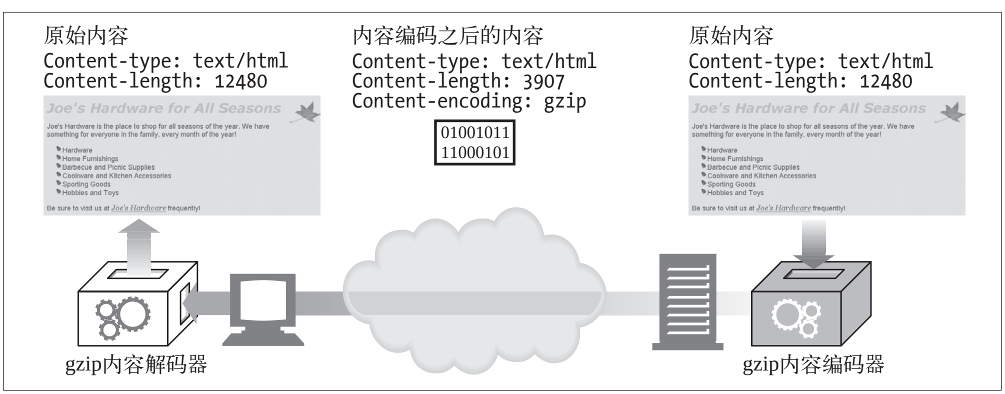


图 15-3 内容编码示例

在这个例子中，通过 gzip 内容编码函数对 HTML 页面处理之后，得到一个更小的、压缩的主体。经过网络发送的是压缩的主体，并打上了 gzip 压缩的标志。接收的客

户端使用 gzip 解码器对实体进行解压缩。

下面给出的响应片段是另一个编码响应的例子（一个压缩的图像）：

```
HTTP/1.1 200 OK
Date: Fri, 05 Nov 1999 22:35:15 GMT
Server: Apache/1.2.4
Content-Length: 6096
Content-Type: image/gif
Content-Encoding: gzip
[...]
```

注意，Content-Type 首部可以且还应当出现在报文中。它说明了实体的原始格式，一旦实体被解码，要显示的时候，可能还是需要该信息才行的。记住，Content-Length 首部现在代表的是*编码之后*的主体长度。

15.5.2 内容编码类型

HTTP 定义了一些标准的内容编码类型，并允许用扩展编码的形式增添更多的编码。由互联网号码分配机构（IANA）对各种编码进行标准化，它给每个内容编码算法分配了唯一的代号。Content-Encoding 首部就用这些标准化的代号来说明编码时使用的算法。

表 15-2 列出了一些常用的内容编码代号。 352

表15-2 内容编码代号

Content-Encoding值	描述
gzip	表明实体采用 GNU zip 编码[a]
compress	表明实体采用 Unix 的文件压缩程序
deflate	表明实体是用 zlib 的格式压缩的[b]
identity	表明没有对实体进行编码。当没有 Content-Encoding 首部时，就默认为这种情况

a：RFC 1952 中说明了 gzip 编码。

b：RFC 1950 和 1951 中讲解了 zlib 格式和 deflate 压缩算法。

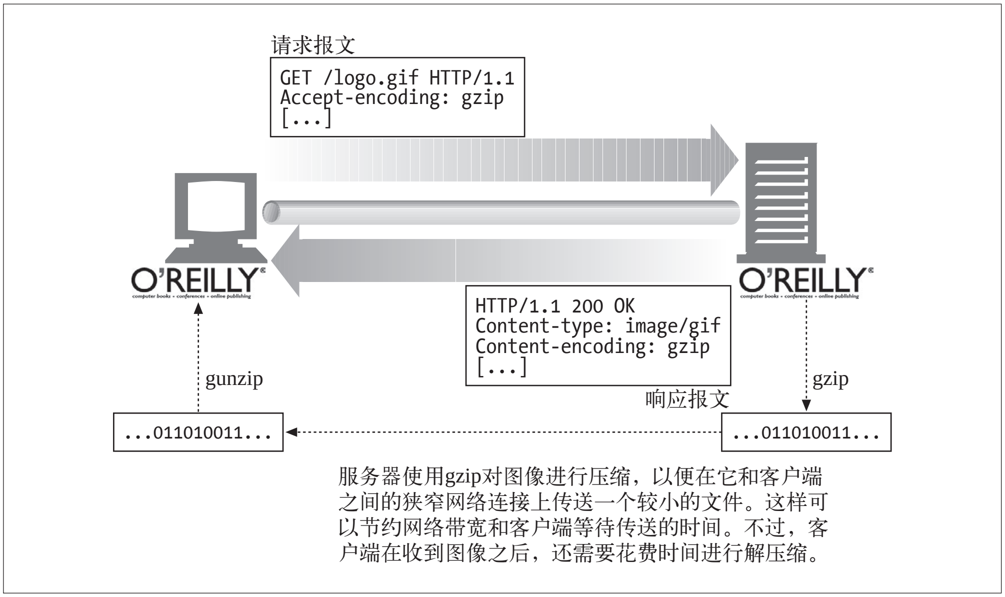
gzip、compress 以及 deflate 编码都是无损压缩算法，用于减少传输报文的大小，不会导致信息损失。这些算法中，gzip 通常是效率最高的，使用最为广泛。

15.5.3 Accept-Encoding首部

毫无疑问，我们不希望服务器用客户端无法解码的方式来对内容进行编码。为了避免服务器使用客户端不支持的编码方式，客户端就把自己支持的内容编码方式

列表放在请求的 Accept-Encoding 首部里发出去。如果 HTTP 请求中没有包含 Accept-Encoding 首部，服务器就可以假设客户端能够接受任何编码方式（等价于发送 Accept-Encoding: *）。

图 15-4 展示了 HTTP 事务中的 Accept-Encoding 首部。



353

图 15-4　展示了 HTTP 事务中的 **Accept-Encoding** 首部

Accept-Encoding 字段包含用逗号分隔的支持编码的列表，下面是一些例子：

```
Accept-Encoding: compress, gzip
Accept-Encoding:
Accept-Encoding: *
Accept-Encoding: compress;q=0.5, gzip;q=1.0
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

客户端可以给每种编码附带 Q（质量）值参数来说明编码的优先级。Q 值的范围从 0.0 到 1.0，0.0 说明客户端不想接受所说明的编码，1.0 则表明最希望使用的编码。“*”表示“任何其他方法”。决定在响应中回送什么内容给客户端是个更通用的过程，而选择使用何种内容编码则是此过程的一部分。第 17 章将详细讨论这个过程，以及 Content-Encoding 首部和 Accept-Encoding 首部。

identity 编码代号只能在 Accept-Encoding 首部中出现，客户端用它来说明相对于其他内容编码算法的优先级。

15.6 传输编码和分块编码

前一节讨论的内容编码，是对报文的主体进行的可逆变换。内容编码是和内容的具体格式细节紧密相关的。例如，你可能会用 gzip 压缩文本文件，但不是 JPEG 文件，因为 JPEG 这类东西用 gzip 压缩的不够好。

本节讨论传输编码。传输编码也是作用在实体主体上的可逆变换，但使用它们是由于架构方面的原因，同内容的格式无关。如图 15-5 所示，使用传输编码是为了改变报文中的数据在网络上传输的方式。

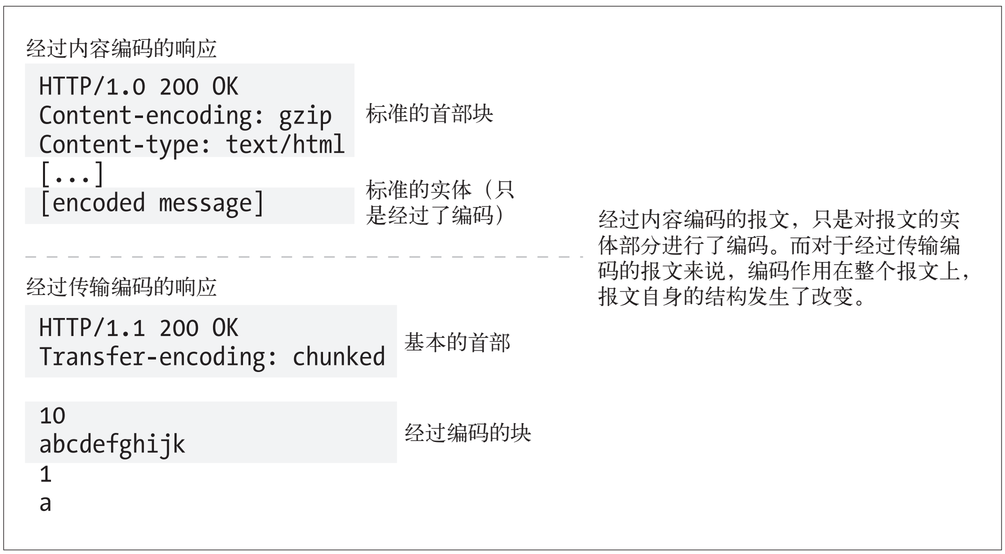


图 15-5　内容编码和传输编码的对比

15.6.1 可靠传输

长久以来，在其他一些协议中会用传输编码来保证报文经过网络时能得到“可靠传输”。在 HTTP 协议中，可靠传输关注的焦点有所不同，因为底层的传输设施已经标准化并且容错性更好。在 HTTP 中，只有少数一些情况下，所传输的报文主体可能会引发问题。其中两种情况如下所述。

- *未知的尺寸*

  如果不先生成内容，某些网关应用程序和内容编码器就无法确定报文主体的最终大小。通常，这些服务器希望在知道大小之前就开始传输数据。因为 HTTP 协议 354
  要求 `Content-Length` 首部必须在数据之前，有些服务器就使用传输编码来发

送数据，并用特别的结束脚注表明数据结束。[7]

- 安全性

  你可以用传输编码来把报文内容扰乱，然后在共享的传输网络上发送。不过，由于像 SSL 这样的传输层安全体系的流行，就很少需要靠传输编码来实现安全性了。

15.6.2 Transfer-Encoding首部

HTTP 协议中只定义了下面两个首部来描述和控制传输编码。

- Transfer-Encoding

  告知接收方为了可靠地传输报文，已经对其进行了何种编码。

- TE

355 用在请求首部中，告知服务器可以使用哪些传输编码扩展。[8]

下面的例子中，请求使用了 TE 首部来告诉服务器它可以接受分块编码（如果是 HTTP/1.1 应用程序的话，这就是必须的）并且愿意接受附在分块编码的报文结尾上的拖挂：

```
GET /new_products.html HTTP/1.1
Host: www.joes-hardware.com
User-Agent: Mozilla/4.61 [en] (WinNT; I)
TE: trailers, chunked
...
```

对它的响应中包含 Transfer-Encoding 首部，用于告诉接收方已经用分块编码对报文进行了传输编码：

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Server: Apache/3.0
...
```

在这个起始首部之后，报文的结构就将发生改变。

传输编码的值都是大小写无关的。HTTP/1.1 规定在 TE 首部和 Transfer-Encoding 首部中使用传输编码值。最新的 HTTP 规范只定义了一种传输编码，就是分块编码。

与 Accept-Encoding 首部类似，TE 首部也可以使用 Q 值来说明传输编码的优先顺序。不过，HTTP/1.1 规范中禁止将分块编码关联的 Q 值设为 0.0。

注 7：尽管可以因陋就简地用关闭连接作为报文结束的信号，但这种方法不能用于持久连接。

注 8：如果这个首部起名叫 Accept-Transfer-Encoding，它的意义就会更直白。

HTTP 将来的扩展可能会推动对更多传输编码的需求。如果真的如此，那分块编码仍应始终作用在其他传输编码之上，这样就保证数据可以像隧道那样“穿透”那些只理解分块编码但不理解其他传输编码的 HTTP/1.1 应用程序。

15.6.3 分块编码

分块编码把报文分割为若干个大小已知的块。块之间是紧挨着发送的，这样就不需要在发送之前知道整个报文的大小了。

要注意的是，分块编码是一种传输编码，因此是报文的属性，而不是主体的属性。本章前面部分讨论过的多部分编码，就是主体的属性，它和分块编码是完全独立的。

1. 分块与持久连接

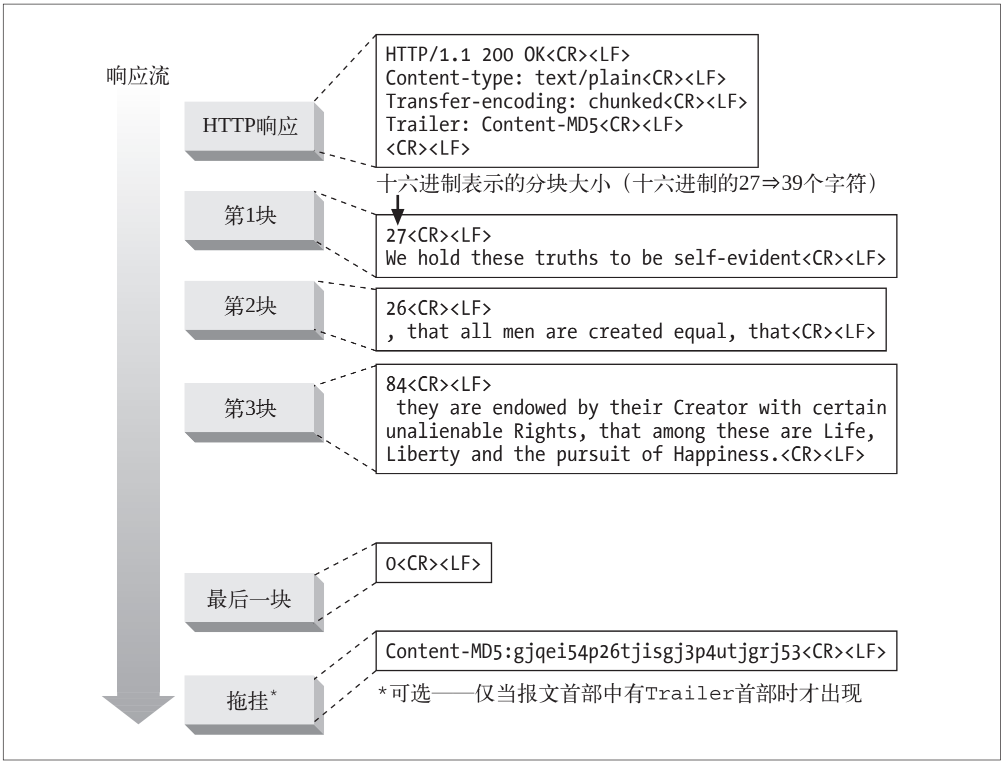
若客户端和服务器之间不是持久连接，客户端就不需要知道它正在读取的主体的长度，而只需要读到服务器关闭主体连接为止。 356

当使用持久连接时，在服务器写主体之前，必须知道它的大小并在 `Content-Length` 首部中发送。如果服务器动态创建内容，就可能在发送之前无法知道主体的长度。

分块编码为这种困难提供了解决方案，只要允许服务器把主体逐块发送，说明每块的大小就可以了。因为主体是动态创建的，服务器可以缓冲它的一部分，发送其大小和相应的块，然后在主体发送完之前重复这个过程。服务器可以用大小为 0 的块作为主体结束的信号，这样就可以继续保持连接，为下一个响应做准备。

分块编码是相当简单的。图 15-6 展示了一个分块编码报文的基本结构。它由起始的 HTTP 响应首部块开始，随后就是一系列分块。每个分块包含一个长度值和该分块的数据。长度值是十六进制形式并将 CRLF 与数据分隔开。分块中数据的大小以字节计算，不包括长度值与数据之间的 CRLF 序列以及分块结尾的 CRLF 序列。最后一个块有点特别，它的长度值为 0，表示“主体结束”。

客户端也可以发送分块的数据给服务器。因为客户端事先不知道服务器是否接受分块编码（这是因为服务器不会在给客户端的响应中发送 `TE` 首部），所以客户端必须做好服务器用 411 Length Required（需要 `Content-Length` 首部）响应来拒绝分块请求的准备。



357 图 15-6　剖析分块编码报文

2. 分块报文的拖挂

如果客户端的 `TE` 首部中说明它可以接受拖挂的话，就可以在分块的报文最后加上拖挂。产生原始响应的服务器也可以在分块的报文最后加上拖挂。拖挂的内容是可选的元数据，客户端不一定需要理解和使用（客户端可以忽略并丢弃拖挂中的内容）。[9]

拖挂中可以包含附带的首部字段，它们的值在报文开始的时候可能是无法确定的（例如，必须要先生成主体的内容）。`Content-MD5` 首部就是一个可以在拖挂中发送的首部，因为在文档生成之前，很难算出它的 MD5。图 15-6 中展示了拖挂的使用方式。报文首部中包含一个 `Trailer` 首部，列出了跟在分块报文之后的首部列表。在 `Trailer` 首部中列出的首部就紧接在最后一个分块之后。

除了 `Transfer-Encoding`、`Trailer` 以及 `Content-Length` 首部之外，其他 HTTP 首部都可以作为拖挂发送。

注 9：`trailer`（拖挂）首部是在最初的分块编码被加入到 HTTP/1.1 规范的草案之后才加入的，因此有些应用程序可能不理解这个首部（或者不理解拖挂），尽管它们声称是兼容 HTTP/1.1 的。

15.6.4　内容编码与传输编码的结合

内容编码与传输编码可以同时使用。例如，图 15-7 中展示了发送方如何用内容编码压缩 HTML 文件，再使用传输编码分块发送。接收方“重构”主体的过程和发送方相反。

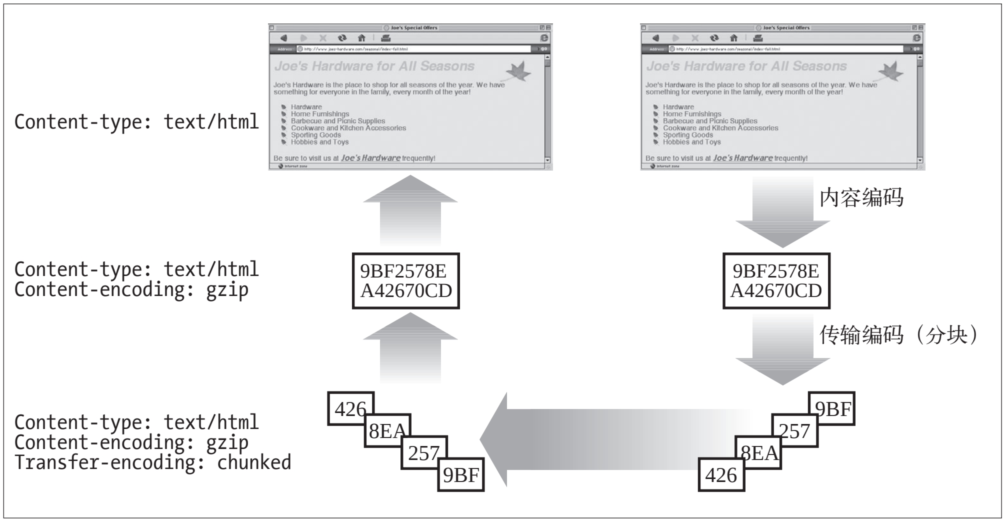


图 15-7　内容编码和传输编码结合

15.6.5　传输编码的规则

对报文主体使用传输编码时，必须遵守以下规则。

- 传输编码集合中必须包括“分块”。唯一的例外是使用关闭连接来结束报文。
- 当使用分块传输编码时，它必须是最后一个作用到报文主体之上的。
- 分块传输编码不能多次作用到一个报文主体上。 358

这些规则使得接收方能够确定报文的传输长度。

传输编码是 HTTP 1.1 版中引入的一个相对较新的特性。实现传输编码的服务器必须特别注意不要把经传输编码后的报文发送给非 HTTP/1.1 的应用程序。同样地，如果服务器收到无法理解的经过传输编码的报文，它应当用 501 Unimplemented 状态码来回复。不过，所有的 HTTP/1.1 应用程序至少都必须支持分块编码。

15.7　随时间变化的实例

网站对象并不是静态的。同样的 URL 会随着时间变化而指向对象的不同版本。以

CNN 的主页为例，同一天里多次访问 http://www.cnn.com，可能每次得到的返回页面都会略有不同。

可以把 CNN 的主页当作一个对象来考虑，其不同版本就可以看作这个对象的不同实例（参见图 15-8）。在图中，客户端多次请求同一个资源（URL），但得到的是该资源的不同实例，因为它是随时间而变化的。在时间（a）和时间（b）具有相同的实例，而在时间（c）则是不同的实例。

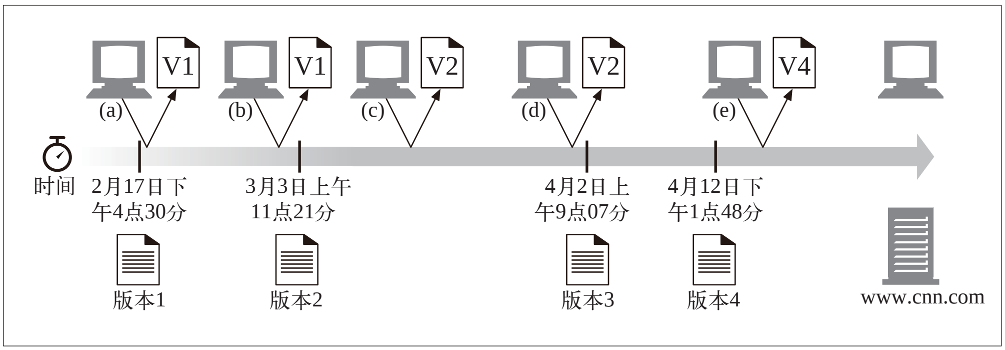


图 15-8　实例是资源在某个时间的“快照”

HTTP 协议规定了称为实例操控（instance manipulations）的一系列请求和响应操作，用以操控对象的实例。两个主要的实例操控方法是范围请求和差异编码。这两种方法都要求客户端能够标识它所拥有（如果有的话）的资源的特定副本，并在一定的条件下请求新的实例。本章后面将讨论这些机制。 359

15.8　验证码和新鲜度

现在再回顾前面的图 15-8。客户端起初没有该资源的副本，因此它发送请求给服务器要求得到一份。服务器用该资源的版本 1 给以响应。客户端现在可以缓存这份副本，但是要缓存多长时间呢？

当文档在客户端“过期”之后（也就是说，客户端不再认为该副本有效），客户端必须从服务器请求一份新的副本。不过，如果该文档在服务器上并未发生改变，客户端也就不需要再接收一次了——继续使用缓存的副本即可。

这种特殊的请求，称为有条件的请求（conditional request），要求客户端使用验证码（validator）来告知服务器它当前拥有的版本号，并仅当它的当前副本不再有效时才要求发送新的副本。让我们进一步详细研究这 3 个关键概念——新鲜度、验证码以及有条件的请求。

15.8.1 新鲜度

服务器应当告知客户端能够将内容缓存多长时间，在这个时间之内就是新鲜的。服务器可以用这两个首部之一来提供这种信息：Expires（过期）和 Cache-Control（缓存控制）。

Expires 首部规定文档"过期"的具体时间——此后就不应当认为它还是最新的。Expires 首部的语法如下：

```
Expires: Sun Mar 18 23:59:59 GMT 2001
```

客户端和服务器为了能正确使用 Expires 首部，它们的时钟必须同步。这并不总是很容易的，因为它们可能都没有运行像 Network Time Protocol（网络时间协议，NTP）这样的时钟同步协议。用相对时间来定义过期的机制会更有用。Cache-Control 首部可以用秒数来规定文档最长使用期——从文档离开服务器之后算起的总计时间。使用期不与时钟同步，因此可以给出更精确的结果。 360

实际上，Cache-Control 首部功能很强大。服务器和客户端都可以用它来说明新鲜度，并且除了使用期或过期时间之外，还有很多指令可用。表 15-3 列出了 Cache-Control 首部的一些指令。

表15-3 Cache-Control首部的指令

指令	报文类型	描述
no-cache	请求	在重新向服务器验证之前，不要返回文档的缓存副本
no-store	请求	不要返回文档的缓存副本。不要保存服务器的响应
max-age	请求	缓存中的文档不能超过指定的使用期
max-stale	请求	文档允许过期（根据服务器提供的过期信息计算），但不能超过指令中指定的过期值
min-fresh	请求	文档的使用期不能小于这个指定的时间与它的当前存活时间之和。换句话说，响应必须至少在指定的这段时间之内保持新鲜
no-transform	请求	文档在发送之前不允许被转换
only-if-cached	请求	只有当文档在缓存中才发送，不要联系原始服务器
public	响应	响应可以被任何服务器缓存
private	响应	响应可以被缓存，但只能被单个客户端访问
no-cache	响应	如果该指令伴随一个首部列表的话，那么内容可以被缓存并提供给客户端，但必须先删除所列出的首部。如果没有指定首部，缓存中的副本在没有重新向服务器验证之前不能提供给客户端
no-store	响应	响应不允许被缓存
no-transform	响应	响应在提供给客户端之前不能做任何形式的修改

（续）

指　　令	报文类型	描　　述
must-revalidate	响应	响应在提供给客户端之前必须重新向服务器验证
proxy-revalidate	响应	共享的缓存在提供给客户端之前必须重新向原始服务器验证。私有的缓存可以忽略这条指令
max-age	响应	指定文档可以被缓存的时间以及新鲜度的最长时间
s-max-age	响应	指定文档作为共享缓存时的最长使用时间（如果有 max-age 指令的话，以本指令为准）。私有的缓存可以忽略本指令

缓存和新鲜度在第 7 章中曾有详细讨论。

15.8.2　有条件的请求与验证码

当请求缓存服务器中的副本时，如果它不再新鲜，缓存服务器就需要保证它有一个新鲜的副本。缓存服务器可以向原始服务器获取当前的副本。但在很多情况下，原始服务器上的文档仍然与缓存中已过期的副本相同。我们在图 15-8b 中看到过这种
361 情况；缓存的副本或许已经过期了，但原始服务器上的内容与缓存的内容仍然相同。如果服务器上的文档和已过期的缓存副本相同，而缓存服务器还是要从原始服务器上取文档的话，那缓存服务器就是在浪费网络带宽，给缓存服务器和原始服务器增加不必要的负载，使所有事情都变慢了。

为了避免这种情况，HTTP 为客户端提供了一种方法，仅当*资源改变*时才请求副本，这种特殊请求称为有条件的请求。有条件的请求是标准的 HTTP 请求报文，但仅当某个特定条件为真时才执行。例如，某个缓存服务器可能发送下面的有条件 GET 报文给服务器，仅当文件 /announce.html 从 2002 年 6 月 29 日（这是缓存的文档最后被作者修改的时间）之后发生改变的情况下才发送它：

```
GET /announce.html HTTP/1.0
If-Modified-Since: Sat, 29 Jun 2002, 14:30:00 GMT
```

有条件的请求是通过以“If-”开头的有条件的首部来实现的。在上面的例子中，有条件的首部是 If-Modified-Since(如果 – 从……之后 – 修改过)。有条件的首部使得方法仅在条件为真时才执行。如果条件不满足，服务器就发回一个 HTTP 错误码。

每个有条件的请求都通过特定的*验证码*来发挥作用。验证码是文档实例的一个特殊属性，用它来测试条件是否为真。从概念上说，你可以把验证码看作文件的序列号、版本号，或者最后发生改变的日期时间。在图 15-8b 中，那个智能的客户端发送给服务器的有条件的验证请求是在说：“我有版本 1，如果这个资源不再是版本 1 就把它发给我。”我们在第 7 章已经讨论过有条件的缓存再验证了，而本章会更仔细地研

究实体验证码的细节。

有条件的首部 If-Modified-Since 测试的是文档实例最后被修改的日期时间，因此我们说最后被修改的日期时间就是验证码。有条件的首部 If-None-Match 测试的是文档的 ETag 值，它是与实体相关联的一个特殊的关键字，或者说是版本识别标记。Last-Modified 和 ETag 是 HTTP 使用的两种主要验证码。表 15-4 中列出了用于有条件请求的 4 种 HTTP 首部。每个有条件的首部之后就是这种首部所用的验证码类型。

表15-4　有条件的请求类型

请求类型	验　证　码	描　　述
If-Modified-Since	Last-Modified	如果在前一条响应的 Last-Modified 首部中说明的时间之后，资源的版本发生变化，就发送其副本
If-Unmodified-Since	Last-Modified	仅在前一条响应的 Last-Modified 首部中说明的时间之后，资源的版本没有变化，才发送其副本
If-Match	ETag	如果实体的标记与前一次响应首部中的 ETag 相同，就发送该资源的副本
If-None-Match	ETag	如果实体的标记与前一次响应首部中的 ETag 不同，就发送该资源的副本

362

HTTP 把验证码分为两类：*弱验证码*（weak validators）和*强验证码*（strong validators）。弱验证码不一定能唯一标识资源的一个实例，而强验证码必须如此。弱验证码的一个例子是对象的大小字节数。有可能资源的内容改变了，而大小还保持不变，因此假想的字节计数验证码与改变是弱相关的。而资源内容的加密校验和（比如 MD5）就是强验证码，当文档改变时它总是会改变。

最后修改时间被当作弱验证码，因为尽管它说明了资源最后被修改的时间，但它的描述精度最大就是 1 秒。因为资源在 1 秒内可以改变很多次，而且服务器每秒可以处理数千个请求，最后修改日期时间并不总能反应变化情况。ETag 首部被当作强验证码，因为每当资源内容改变时，服务器都可以在 ETag 首部放置不同的值。版本号和摘要校验和也是很好的 ETag 首部候选，但它们不能带有任意的文本。ETag 首部很灵活，它可以带上任意的文本值（以标记的形式），这样就可以用来设计出各种各样的客户端和服务器验证策略。

有时候，客户端和服务器可能需要采用不那么精确的实体标记验证方法。例如，某服务器可能想对一个很大、被广泛缓存的文档进行一些美化修饰，但不想在缓存服务器再验证时产生很大的传输流量。在这种情况下，该服务器可以在标记前面加上“W/”前缀来广播一个“弱”实体标记。对于弱实体标记来说，只有当关联的实体

在语义上发生了重大改变时，标记才会变化。而强实体标记则不管关联的实体发生了什么性质的变化，标记都一定会改变。

下面的例子展示了客户端如何用弱实体标记向服务器请求再验证。服务器仅当文档的内容从版本 4.0 算起发生了显著变化时，才返回主体：

```
GET /announce.html HTTP/1.1
If-None-Match: W/"v4.0"
```

概括一下，当客户端多次访问同一个资源时，首先需要判断它当前的副本是不是仍然新鲜。如果不再新鲜，它们就必须从服务器获取最新的版本。为了避免在资源没有改变的情况下收到一份相同的副本，客户端可以向服务器发送有条件的请求，说明能唯一标识客户端当前副本的验证码。只在资源和客户端的副本不同的情况下服务器才会发送其副本。更多关于缓存再验证的细节，请回顾 7.7 节。

15.9 范围请求

关于客户端如何要求服务器只在资源的客户端副本不再有效的情况下才发送其副本，我们已经清楚地理解了。HTTP 还进一步锦上添花：它允许客户端实际上只请求文
363 档的一部分，或者说某个范围。

假设你正通过慢速的调制解调器连接下载最新的热门软件，已经下了四分之三，忽然因为一个网络故障，连接中断了。你已经为等待下载完成耽误了很久，而现在被迫要全部重头再来，祈祷着别再发生这样的倒霉事了。

有了范围请求，HTTP 客户端可以通过请求曾获取失败的实体的一个范围（或者说一部分），来恢复下载该实体。当然这有一个前提，那就是从客户端上一次请求该实体到这次发出范围请求的时段内，该对象没有改变过。例如：

```
GET /bigfile.html HTTP/1.1
Host: www.joes-hardware.com
Range: bytes=4000-
User-Agent: Mozilla/4.61 [en] (WinNT; I)
...
```

在本例中，客户端请求的是文档开头 4000 字节之后的部分（不必给出结尾字节数，因为请求方可能不知道文档的大小）。在客户端收到了开头的 4000 字节之后就失败的情况下，可以使用这种形式的范围请求。还可以用 `Range` 首部来请求多个范围（这些范围可以按任意顺序给出，也可以相互重叠）。例如，假设客户端同时连接到多个服务器，为了加速下载文档而从不同的服务器下载同一个文档的不同部分。对于客户端在一个请求内请求多个不同范围的情况，返回的响应也是单个实体，它有

一个多部分主体及 Content-Type：multipart/byteranges 首部。

并不是所有服务器都接受范围请求，但很多服务器可以。服务器可以通过在响应中包含 Accept-Ranges 首部的形式向客户端说明可以接受的范围请求。这个首部的值是计算范围的单位，通常是以字节计算的。[10] 例如：

```
HTTP/1.1 200 OK
Date: Fri, 05 Nov 1999 22:35:15 GMT
Server: Apache/1.2.4
Accept-Ranges: bytes
...
```

图 15-9 展示了涉及范围请求的一系列 HTTP 事务的例子。

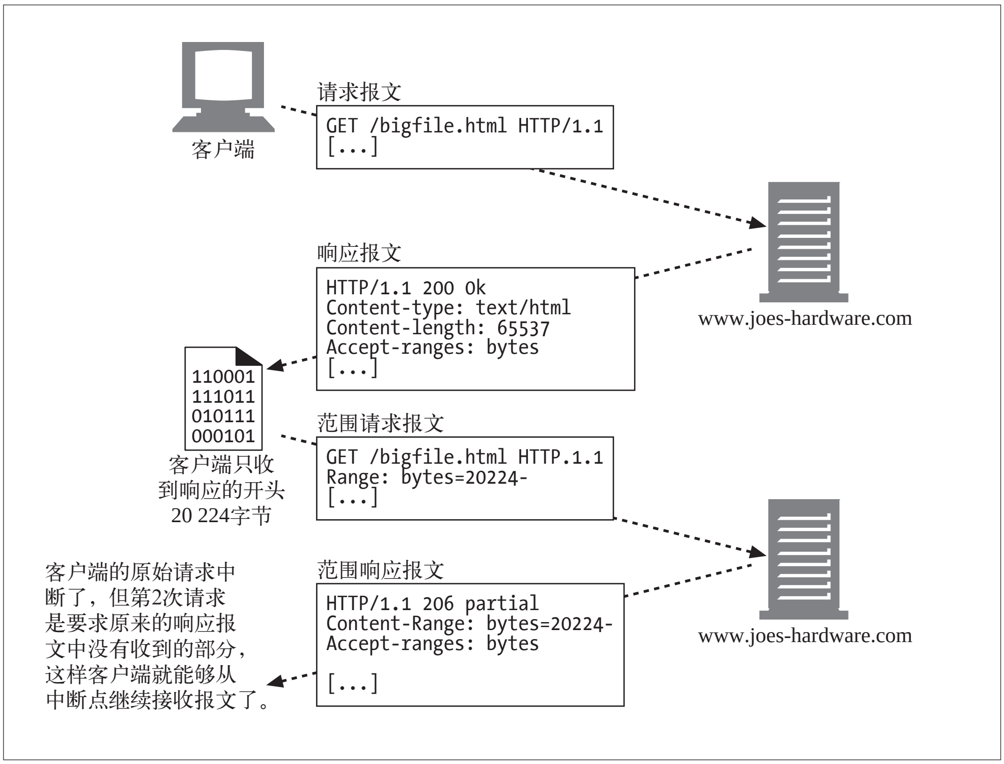


图 15-9　实体范围请求示例

Range 首部在流行的点对点（Peer-to-Peer，P2P）文件共享客户端软件中得到广泛应用，它们从不同的对等实体同时下载多媒体文件的不同部分。

注 10：HTTP/1.1 规范中只定义了 bytes 记号，但服务器和客户端的具体实现可以用它们自己认定的单位来衡量或切分实体。

注意，范围请求也属于一类实例操控，因为它们是在客户端和服务器之间针对特定的对象实例来交换信息的。也就是说，客户端的范围请求仅当客户端和服务器拥有
364 文档的同一个版本时才有意义。

15.10 差异编码

我们曾把网站页面的不同版本看作页面的不同实例。如果客户端有一个页面的已过期副本，就要请求页面的最新实例。如果服务器有该页面更新的实例，就要把它发给客户端，哪怕页面上只有一小部分发生了改变，也要把完整的新页面实例发给客户端。

若改变的地方比较少，与其发送完整的新页面给客户端，客户端更愿意服务器只发送页面发生改变的部分，这样就可以更快地得到最新的页面。差异编码是 HTTP 协议的一个扩展，它通过交换对象改变的部分而不是完整的对象来优化传输性能。差异编码也是一类实例操控，因为它依赖客户端和服务器之间针对特定的对象实例来交换信息。RFC 3229 描述了差异编码。

图 15-10 更清楚地展示了差异编码的结构，包括请求、生成、接收和装配文档的全过程。客户端必须告诉服务器它有页面的哪个版本，它愿意接受页面最新版的差异（delta），它懂得哪些将差异应用于现有版本的算法。服务器必须检查它是否有这个
365 页面的客户端现有版本，计算客户端现有版本与最新版之间的差异（有若干算法可以计算两个对象之间的差异）。然后服务器必须计算差异，发送给客户端，告知客户端所发送的是差异，并说明最新版页面的新标识（`ETag`），因为客户端将差异应用于其老版本之后就会得到这个版本。

客户端在 `If-None-Match` 首部中使用的是它所持有页面版本的唯一标识，这个标
366 识是服务器之前响应客户端时在 `ETag` 首部中发送的。客户端是在对服务器说：“如果你那里页面的最新版本标识和这个 `ETag` 不同，就把这个页面的最新版本发给我。”如果只有 `If-None-Match` 首部，服务器将会把该页面的最新版本完整地发给客户端。（假设最新版和客户端持有的版本不同。）

不过，如果客户端想告诉服务器它愿意接受该页面的差异，只要发送 `A-IM` 首部就可以了。`A-IM` 是 Accept-Instance-Manipulation（接受实例操控）的缩写。形象比喻的话，客户端相当于这样说：“哦对了，我能接受某些形式的实例操控，如果你会其中一种的话，就不用发送完整的文档给我了。”在 `A-IM` 首部中，客户端会说明它知道哪些算法可以把差异应用于老版本而得到最新版本。服务端发送回下面这些内容：一个特殊的响应代码——226 IM Used，告知客户端它正在发送的是所请求对象

的实例操控，而不是那个完整的对象自身；一个 `IM`（Instance-Manipulation 的缩写）首部，说明用于计算差异的算法；新的 `ETag` 首部和 `Delta-Base` 首部，说明用于计算差异的基线文档的 `ETag`（理论上，它应该和客户端之前请求里的 `If-None-Match` 首部中的 `ETag` 相同！）。表 15-5 中总结了差异编码使用的首部。

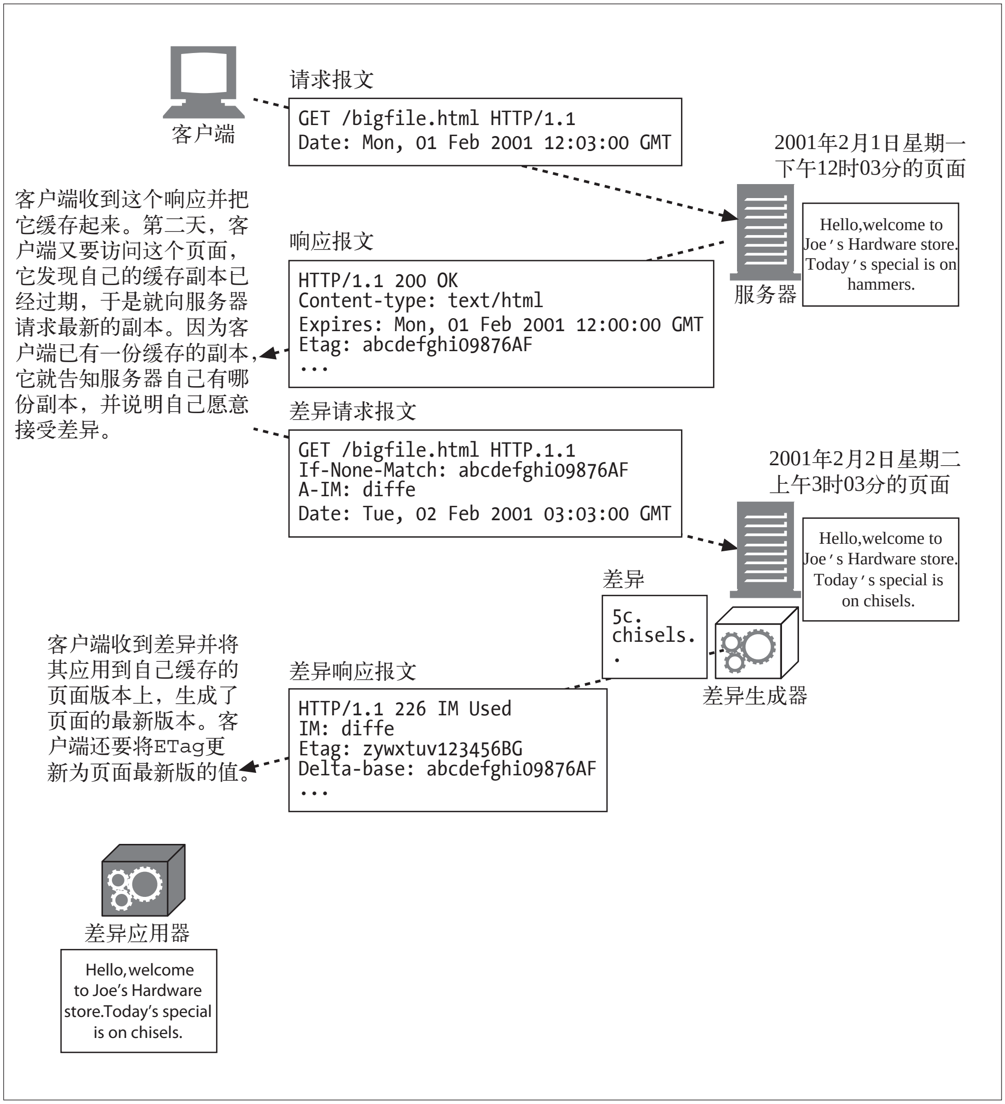


图 15-10　差异编码的结构

表15-5　差异编码所用的首部

首　部	描　述
ETag	文档每个实例的唯一标识符。由服务器在响应中发送；客户端在后继请求的 If-Match 首部和 If-None-Match 首部中可以使用它
If-None-Match	客户端发送的请求首部，当且仅当客户端的文档版本与服务器不同时，才向服务器请求该文档
A-IM	客户端请求首部，说明可以接受的实例操控类型
IM	服务器响应首部，说明作用在响应上的实例操控的类型。当响应代码是 226 IM Used 时，会发送这个首部
Delta-Base	服务器响应首部，说明用于计算差异的基线文档的 ETag 值（应当与客户端请求中的 If-None-Match 首部里的 ETag 相同）

实例操控、差异生成器和差异应用器

客户端可以使用 A-IM 首部说明可以接受的一些实例操控的类型。服务器在 IM 首部中说明使用的是何种实例操控。不过到底哪些实例操控类型是可接受的呢？它们又是做什么的呢？表 15-6 中列出了一些在 IANA 注册的实例操控类型。

表 15-6　在IANA注册的实例操控类型

	类　型	说　明
	vcdiff	用 vcdiff 算法计算差异[a]
	diffe	用 Unix 系统的 diff-e 命令计算差异
367	gdiff	用 gdiff 算法计算差异[b]
	gzip	用 gzip 算法压缩
	deflate	用 deflate 算法压缩
	range	用在服务器的响应中，说明响应是针对范围选择得到的部分内容
	identity	用在客户端请求中的 A-IM 首部中，说明客户端愿意接受恒等实例操控

a：因特网草案 draft-korn-vcdiff-01 中描述了 vcdiff 算法。该规范在 2002 年初期由 IESG（Internet Engineering Steering Group，因特网工程指导组）批准，将很快以 RFC 的形式发布。（译注：vcdiff 的规范由 RFC3284 发布。）

b：http://www.w3.org/TR/NOTE-gdiff-19970901.html 描述了 gdiff 算法。

图 15-10 中，服务器侧的“差异生成器”根据基线文档和该文档的最新实例，用客户端在 A-IM 首部中指明的算法计算它们之间的差异。客户端侧的“差异应用器”得到差异，将其应用于基线文档，得到文档的最新实例。例如，如果产生差异的算法是 Unix 系统的 diff-e 命令，客户端就可以用 Unix 系统中的文本编辑器 ed 提供的功能来应用差异，因为 diff-e <file1> <file2> 产生了一系列 ed 命令来把 <file1>

转化为 `<file2>`。ed 是一个非常简单的编辑器，支持一些命令。在图 15-10 的例子中，5c 说明要删除基线文档的第 5 行，而 `chisels.<cr>.` 说明要添加 chisels.，就这么简单。对于更大的改动，会产生更复杂的指令。Unix 系统的 `diff-e` 算法是对文件进行逐行比较的，这对于文本文件没问题，但并不适合二进制文件。vcdiff 算法更强大，对于非文本文件也适用，并且产生的差异比 `diff-e` 要小。

差异编码的规范中详细定义了 `A-IM` 和 `IM` 首部的格式。在这里，我们只要知道这些首部中可以说明多个实例操控（并可以带有相关的质量值）就够了。在返回给客户端之前，文档可以经过多种实例操控，这样可以获得最大程度的压缩。例如，用 vcdiff 算法产生的差异随后可以再用 gzip 算法压缩。于是服务器的响应中就含有 `IM:vcdiff, gzip` 首部。客户端应当先对内容进行 gunzip，再把得到的差异应用到自己的基线页面上，这样才能生成最终的文档。

差异编码可以减少传输次数，但实现起来可能比较麻烦。设想一下页面改动频繁，而且有很多不同的人都在访问的情形。支持差异编码的服务器必须保存页面随时间变化的所有不同版本，这样才能指出最新版本与所请求的客户端持有的任意版本之间的差异。（如果文档变化频繁，而且有很多客户端都在请求文档，那它们就会获得 368
文档的不同实例。随后当它们再向服务器发起请求时，它们将请求它们所持有的版本与最新版本之间的差异。为了能够只向它们发送变化的部分，服务器必须保存所有客户端曾经持有过的版本。）要降低提交文档时的延迟时间，服务器必须增加磁盘空间来保存文档的各种旧的实例。实现差异编码所需的额外磁盘空间可能很快就会将减少传输量获得的好处抵消掉。

15.11 更多信息

关于实体和编码方面的更多信息，请参考以下资源。

- http://www.ietf.org/rfc/rfc2616.txt
  RFC 2616，也就是 HTTP/1.1 版的规范，是实体主体管理和编码方面的主要参考。

- http://www.ietf.org/rfc/rfc3229.txt
  RFC 3229，“Delta Encoding in HTTP”（“HTTP 中的差异编码”），说明了如何通过扩展 HTTP/1.1 来支持差异编码。

- *Introduction to Data Compression*[11]（《数据压缩导论》）
  这本书的作者是 Khalid Sayood，出版商为 Morgan Kaufmann Publishers。该书介绍了几种 HTTP 内容编码支持的压缩算法。

注 11：本书影印版由人民邮电出版社出版。（编者注）

- http://www.ietf.org/rfc/rfc1521.txt

  RFC 1521，“Multipurpose Internet Mail Extensions, Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies”（“多用途因特网邮件扩展，第一部分：规定和描述因特网报文主体格式的机制”），描述了 MIME 主体的格式。这份参考材料很有用，因为 HTTP 从 MIME 中借用了大量内容。设计这份文档的目的，就是为了提供在单一报文中包含多个对象的各种设施，比如用 US-ASCII 之外的字符集来表示主体文本，表示多种字体格式的文本消息以及表示非文本类的信息，比如图像和声音片段等。

- http://www.ietf.org/rfc/rfc2045.txt

  RFC 2045，“Multipurpose Internet Mail Extensions, Part One: Format of Internet Message Bodies”（“多用途因特网邮件扩展，第一部分：因特网报文主体的格式”），规定了用来描述 MIME 格式报文结构的各种首部，其中许多都和 HTTP 中的用法类似或相同。

- http://www.ietf.org/rfc/rfc1864.txt

  RFC 1864，“The `Content-MD5` Header Field”（“`Content-MD5` 首部字段”），提供了用 `Content-MD5` 首部字段来做报文完整性检查的行为及用途方面的一些历史细节。

- http://www.ietf.org/rfc/rfc3230.txt

  RFC 3230，“Instance Digests in HTTP”（“HTTP 中的实例摘要”），描述了对

  369 HTTP 实体摘要处理的改进，解决了 `Content-MD5` 中存在的各种问题。

第16章

国际化

每天有上亿的人用数百种语言写着各种文档。为了真正实现万维网的目标，HTTP 要能够传输和处理用多种语言和字母表编写的国际性文档。

本章涵盖网站国际化方面的两个主要问题：*字符集编码*（character set encoding）和*语言标记*（language tag）。HTTP 应用程序使用字符集编码请求和显示不同字母表中的文本，它们使用语言标记根据用户所理解的语言来说明并限制内容。而在本章的最后将讨论多语言 URI 和日期格式。

本章主要内容：

- 讲解 HTTP 如何与多语言字母表的方案和相关标准进行交互；
- 快速概览术语、技术和标准，以帮助 HTTP 编程人员正确理解（熟悉字符编码的读者可以跳过本节）；
- 解释对各种语言的标准命名系统，以及标准化的语言标记如何描述和选择内容；
- 概述国际性的 URI 要遵循的规则和注意事项；
- 简要讨论日期格式和其他国际化方面的问题。

16.1　HTTP对国际性内容的支持

HTTP 报文中可以承载以任何语言表示的内容，就像它能承载图像、影片，或任何类型的媒体那样。对 HTTP 来说，实体主体只是二进制信息的容器而已。

为了支持国际性的内容，服务器需要告知客户端每个文档的字母表和语言，这样客
370 户端才能正确地把文档中的信息解包为字符并把内容呈现给用户。

服务器通过 HTTP 协议的 `Content-Type` 首部中的 `charset` 参数和 `Content-Language` 首部告知客户端文档的字母表和语言。这些首部描述了实体主体的“信息盒子”里面装的是什么，如何把内容转换成合适的字符以便显示在屏幕上以及里面的词语表示的是哪种语言。

同时，客户端需要告知服务器用户理解何种语言，浏览器上安装了何种字母表编码算法。客户端发送 `Accept-Charset` 首部和 `Accept-Language` 首部，告知服务器它理解哪些字符集编码算法和语言以及其中的优先顺序。

下面的 HTTP 报文中的这些 `Accept` 首部可能是母语为法语的人发出的。他喜欢使用母语，但也会说一点儿英语，他的浏览器支持 iso-8859-1 西欧字符集编码和 UTF-8 Unicode 字符集编码：

```
Accept-Language: fr, en;q=0.8
Accept-Charset: iso-8859-1, utf-8
```

参数“q=0.8”是质量因子（quality factor），说明英语的优先级（0.8）比法语低（默认值是 1.0）。

16.2 字符集与HTTP

现在我们进入主题，开始研究网站国际化中最重要（且令人困惑）的方面——各国的字母表和它们的字符集编码。

Web 字符集标准很有些令人迷惑。由于必须阅读很多标准文档，其中术语复杂且不一致，再加上对外语不太熟悉，很多人首次尝试编写国际化的网站软件时，都被搞糊涂了。本节和下一节应该能让读者更容易地学会在 HTTP 中使用字符集。

16.2.1 字符集是把字符转换为二进制码的编码

HTTP 字符集的值说明如何把实体内容的二进制码转换为特定字母表中的字符。每个字符集标记都命名了一种把二进制码转换为字符的算法（反之亦然）。字符集标记在由 IANA 维护（参见 http://www.iana.org/assignments/character-sets）的 MIME 字符集注册机构进行了标准化。附录 H 中概述了其中的很多字符集。

下面的 `Content-Type` 首部告知接收者，传输的内容是一份 HTML 文件，用 `charset` 参数告知接收者使用 iso-8859-6 阿拉伯字符集的解码算法把内容中的二进制码转换为字符： 371

```
Content-Type: text/html; charset=iso-8859-6
```

iso-8859-6 的编码算法把 8 位值域映射为拉丁字母和阿拉伯字母，以及数字，标点和其他符号[1]。例如，在图 16-1 中，突出显示的二进制码的值是 225，它在 iso-8859-6 中被映射到阿拉伯字母“FEH”（读音类似英语字母 F）。

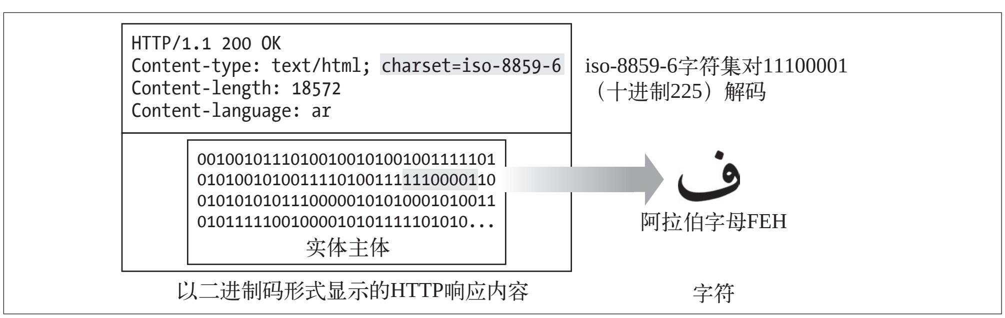


图 16-1 `charset` 参数告知客户端如何把内容中的二进制码转换为字符

注 1：与汉语、日语不同的是，阿拉伯语中只有 28 个字符。8 位空间有 256 个不同的值，足以容纳拉丁字符、阿拉伯字符以及其他符号。

有些字符编码（比如 UTF-8 和 iso-2022-jp）更加复杂，它们是可变长（variable-length）编码，也就是说每个字符的位数都是可变的。这种类型的编码允许使用额外的二进制位表示拥有大量字符的字母表（比如汉语和日语），仅用较少的二进制位来表示标准的拉丁字符。

16.2.2 字符集和编码如何工作

现在我们来看看字符集和编码到底做了什么。

我们想把文档中的二进制码转换为字符以便显示在屏幕上。但由于有很多不同的字母表，也有很多不同的方法把字符编码成二进制码（这些方法各有优缺点），我们需要一种标准方法来描述并应用把二进制码转换为字符的解码算法。

把二进制码转换为字符要经过两个步骤，如图 16-2 所示。

- 在图 16-2a 中，文档中的二进制码被转换成字符代码，它表示了特定编码字符集中某个特定编号的字符。在这个例子里，解码后的字符代码是数字编号 225。
- 在图 16-2b 中，字符代码用于从编码的字符集中选择特定的元素。在 iso-8859-6 中，值 225 对应阿拉伯字母“FEH”。在步骤 a 和 b 中使用的算法取决于 MIME 的 `charset` 标记。

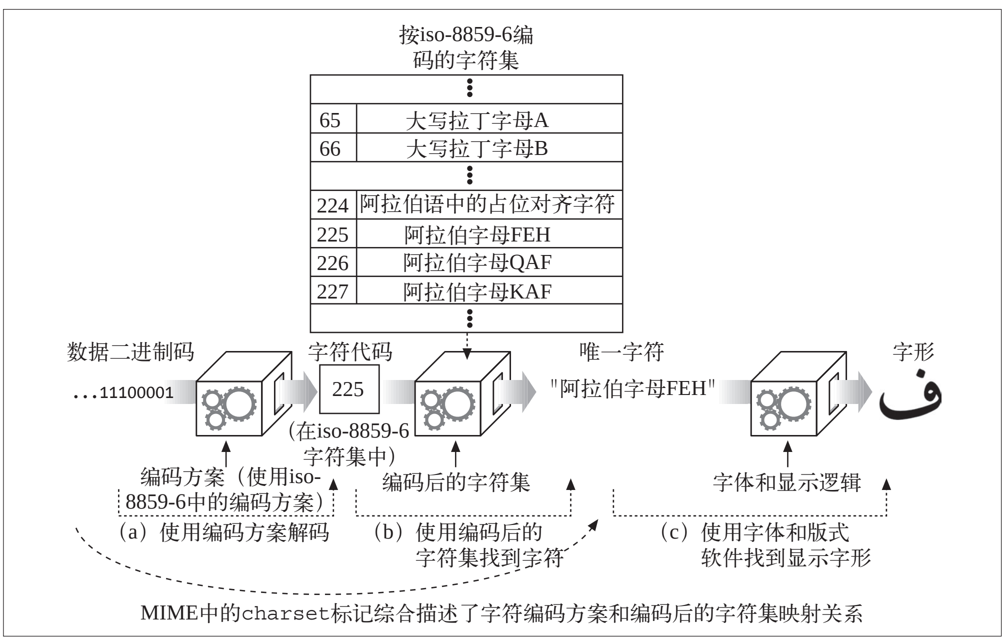


图 16-2　HTTP 协议中的 `charset` 是字符编码方案和编码后字符集的组合

国际化字符系统的关键目标是把语义（字母）和表示（图形化的显示形式）隔离开来。HTTP 只关心字符数据和相关语言及字符集标签的传输。字符形状的显示 372
是由用户的图形显示软件（包括浏览器、操作系统、字体等）完成的，如图 16-2c 所示。

16.2.3　字符集不对，字符就不对

如果客户端使用了错误的字符集参数，客户端就会显示一些奇怪的错乱字符。假设浏览器从主体中获得值 225（二进制为 11100001）。

- 如果浏览器认为主体是用 iso-8859-1 西欧字符编码的，它将会显示带有重音符号的小写拉丁字母“a”：

á

- 如果浏览器使用 iso-8859-6 阿拉伯编码，它将会显示阿拉伯字母“FEH”：

ف

- 如果浏览器使用 iso-8859-7 希腊编码，它将会显示小写的希腊字母“Alpha”： 373

α

- 如果浏览器使用 iso-8859-8 希伯来编码，它将会显示希伯来字母“BET”：

ב

16.2.4　标准化的MIME `charset`值

特定的字符编码方案和特定的已编码字符集组合成一个 MIME 字符集（MIME charset）。HTTP 在 `Content-Type` 和 `Accept-Charset` 首部中使用标准化的 MIME `charset` 标记。MIME `charset` 的值都会在 IANA 注册。[2] 表 16-1 列出了文档和浏览器所使用的一些 MIME `charset` 编码方案。更完整的列表参见附录 H。

注 2：请从 http://www.iana.org/numbers.htm 获取注册的 charset 值的完整列表。

表16-1　MIME `charset` 编码标记

MIME `charset` 值	描　述
us-ascii	这是个著名的字符编码，在 1968 年就已标准化，称为 ANSI_X3.4-1968。它也称为 ASCII，但最好还是加上“US”前缀，因为 ISO 646 中有某些国际化的变体，它们修改了一些字符。US-ASCII 把 7 位数值映射到 128 个字符上。最高位未使用
iso-8859-1	iso-8859-1 是对 ASCII 的 8 位扩展，以支持西欧的多种语言。它使用了最高位以包含更多西欧字符，而保持 ASCII 的编码部分（0 ~ 127）没有变。它也称为 iso-latin-1，或简称为 Latin1
iso-8859-2	对 ASCII 扩展以包括中欧和东欧语言中的字符，包括捷克、波兰、罗马尼亚。它也称为 iso-latin-2
iso-8859-5	对 ASCII 扩展以包括斯拉夫语字符，使用这些字符的语言包括俄语、塞尔维亚语和保加利亚语
iso-8859-6	对 ASCII 扩展以包括阿拉伯语字符。因为阿拉伯语字符的显示形状会随它在单词中的位置而变化，阿拉伯语的显示引擎需要分析上下文来为每个字符生成正确的形状
iso-8859-7	对 ASCII 扩展以包括现代希腊语字符。以前称为 ELOT-928 或 ECMA-118:1986
iso-8859-8	对 ASCII 扩展以包括希伯来语和意第绪语（这两种语言都是犹太人所用的）的字符
iso-8859-15	更新了 iso-8859-1，用遗漏的法语和芬兰语字母替换了一些不太常用的标点符号和分数符号，并用新的欧元符号替换国际货币符号。这种字符集简称为 Latin0，可能将来会替代 iso-8859-1，作为欧洲的首选默认字符集
iso-2022-jp	iso-2022-jp 是在日语的电子邮件和网页内容中广泛使用的编码。它是一种变长编码方案，支持用单字节表示 ASCII 字符，但使用 3 字符的模态转义序列在 3 种日语字符集中切换
euc-jp	euc-jp 是遵循 ISO 2022 的变长编码，它用显式的二进制码模式来标识每个字符，不需要模态及转义序列。它使用单字节、2 字节以及 3 字节的序列来标识多个日语字符集中的字符
Shift_JIS	该编码起初是由微软公司开发的，有时称为 SJIS 或 MS Kanji。出于保持历史兼容性方面的原因，它有点儿复杂，并且不能映射所有的字符，不过它还是用的很 374 普遍
koi8-r	KOI8-R 是为俄语设计的流行的 8 位因特网字符集编码，在 IETF RFC 1489 中定义。 这些大写字母是 Code for Information Exchange, 8 bit, Russian（俄语 8 位信息交换代码）的首字母缩略形式
utf-8	UTF-8 是一种用来表示 UCS（Unicode）的常用变长字符编码方案，UCS 的意思是 Universal Character Set of the world’s characters（世界字符统一字符集）。UTF-8 使用变长的编码来表示字符代码值，每个字符使用 1 ~ 6 个字节。UTF-8 的主要特点之一就是保持对普通的 7 位 ASCII 文本的后向兼容性
windows-1252	微软公司把它编码后的字符集称为 code page（代码页）。Windows 的代码页 1252（也称为 CP1252 或 WinLatin1）是对 iso-8859-1 的扩展

16.2.5 Content-Type首部和Charset首部以及META标志

Web 服务器通过在 `Content-Type` 首部中使用 `charset` 参数把 MIME 字符集标记发送给客户端：

```
Content-Type: text/html; charset=iso-2022-jp
```

如果没有显式地列出字符集，接收方可能就要设法从文档内容中推断出字符集。对于 HTML 内容来说，可以在描述 `charset` 的 `<META HTTP-EQUIV="Content-Type">` 标记中找到字符集。

例 16-1 中展示了 HTML META 标记如何把字符集设置为日语编码 iso-2022-jp。如果文档不是 HTML 类型，或其中没有 `META Content-Type` 标记，软件可以设法扫描实际的文本，看看能否找出语言和编码的常见模式，以此推断字符编码。

例 16-1　可以在 HTML META 标记中规定字符编码

```
<HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-2022-jp">
    <META LANG="jp">
    <TITLE>A Japanese Document</TITLE>
</HEAD>
<BODY>
...
```

如果客户端无法推断出字符编码，就假定使用的是 iso-8859-1。

16.2.6 Accept-Charset首部

在过去的几十年间，人们开发了成千上万种字符编解码方法。大多数客户端不可能支持所有这些不同的字符编码和映射系统。

HTTP 客户端可以使用 `Accept-Charset` 请求首部来明确告知服务器它支持哪些字符系统。`Accept-Charset` 首部的值列出了客户端支持的字符编码方案。例如，下面的 HTTP 请求首部表明，客户端接受西欧字符系统 iso-8859-1 和 UTF-8 变长的 Unicode 兼容系统。服务器可以随便选择这两种字符编码方案之一来返回内容。 375

```
Accept-Charset: iso-8859-1, utf-8
```

注意，没有 `Content-Charset` 这样的响应首部和 `Accept-Charset` 请求首部匹配。为了和 MIME 标准兼容，响应的字符集是由服务器通过 `Content-Type` 响应首部的 `charset` 参数带回来的。不对称真是太糟了，不过需要的信息倒是都有了。

16.3 多语言字符编码入门

前一节描述了客户端和服务器是如何分别在 HTTP 的 `Accept-Charset` 首部和 `Content-Type` 首部的 `charset` 参数中携带字符编码信息的。对于工作中要开发大量国际化应用的 HTTP 程序员来说，为了能理解技术规范和相应的实现软件，需要深入地理解多语言字符系统。

由于术语很复杂且不一致，学习多语言字符系统不太容易。常常需要阅读标准文档，而且我们可能对工作涉及的那些语言还不太熟悉。本节是对字符系统及其标准的概览。如果读者对字符编码很熟悉，或者对这部分细节不感兴趣，可以直接跳到 16.4 节。

16.3.1 字符集术语

以下是应当了解的电子化字符系统的 8 个术语。

- *字符*

  字符是指字母、数字、标点、表意文字（比如汉语）、符号，或其他文本形式的书写“原子”。由统一字符集（Universal Character Set，UCS, 它的非正式的名字是 Unicode[3]）首创，为多种语言中的很多字符开发了一系列标准化的文本名称，它们常用来便捷地命名字符，而且不会与其他字符冲突。[4]

- *字形*

  描述字符的笔画图案或唯一的图形化形状。如果一个字符有多种不同的写法，就有多个字形（参见图 16-3）。

- *编码后的字符*

  分配给字符的唯一数字编号，这样我们就可以操作它了。

- *代码空间*

  376 计划用于字符代码值的整数范围。

- *代码宽度*

  每个（固定大小的）字符代码所用的位数。

- *字符库*

  特定的工作字符集（全体字符的一个子集）。

注 3：Unicode 是一个以 UCS 为基础而成立的商业化联合组织，致力推广商业产品。

注 4：字符的名称看起来类似 LATIN CAPITAL LETTER S 和 ARABIC LETTER QAF 的形式。

- 编码后的字符集

  组成字符库（从全球的字符中选出若干字符）的已编码字符集，并为每个字符分配代码空间中的一个代码。换句话说，它把数字化的字符代码映射为实际的字符。

- 字符编码方案

  把数字化的字符代码编码成一系列二进制码（并能相应地反向解码）的算法。字符编码方案可用来减少识别字符所需要的数据总量（压缩）、解决传输限制、统一重叠编码字符集。

16.3.2 字符集的命名很糟糕

从技术上说，MIME 中的 `charset` 标记（用在 `Content-Type` 首部的 `charset` 参数中和 `Accept-Charset` 首部中）描述的压根就不是字符集。MIME 中的 charset 值所命名的是把数据位映射为唯一的字符的一整套算法。它是字符编码方案（character encoding scheme）和编码后的字符集（coded character set）这两种概念的组合（参见图 16-2）。

因为关于字符编码方案和编码后的字符集方面的标准都已经发布过了，所以，这个术语的使用是很草率的，很容易引起混淆[5]。下面是 HTTP/1.1 的作者们对于他们如何使用这些术语的介绍（在 RFC2616 中）。

> 术语“字符集”在本文档中是指一种方法，它可以把一系列8位字节转换为一系列字符。注意：术语“字符集”经常被称为“字符编码”。但由于HTTP和MIME共享同样的注册信息，术语也能共享是很重要的。

IETF 在 RFC 2277 中也采用了非标准的术语。

> 本文档中使用术语“字符集”来表示一组把一系列 8 位字节转换为一系列字符的规则的集合，比如编码后的字符集与字符编码方案的组合。这与 MIME 的“`charset=`”参数中标识符的用法相同，并且已在 IANA 的字符集注册表中注册。（注意这不是在其他标准主体，比如在国际标准化组织 ISO 中使用的术语）。

注 5：更糟糕的是，MIME 中的 `charset` 标记经常会从特定的编码后字符集的名称或编码方案的名称里面选取。例如，iso-8859-1 是一个编码后字符集（它为一个包含 256 个欧洲字符的集合分配了数字化的代码），但 MIME 用 `charset` 值 iso-8859-1 来表示一种 8 位的、对编码后的字符集恒等的编码。这种不精确的术语并不是致命的问题，但在阅读标准文档的时候，需要对其假设用法保持清醒的头脑。

因此，在阅读标准文档的时候，要保持清醒，这样才能确切地知道它所定义的到底是什么。现在我们已经将相关的术语介绍完了，接下来更仔细地研究一下字符、字
377 形、字符集以及字符编码。

16.3.3 字符

字符是书写的最基本的构建单元。字符可以表示字母、数字、标点、表意符号（比如在汉语中）、数学符号，或其他书写的基本单元。

字符和字体以及风格无关。图 16-3 显示了同一个字符（UCS 中的命名是 LATIN SMALL LETTER A）的若干变体。尽管它们的笔画图案和风格有很大的不同，但母语是西欧语言的读者都能立刻辨认出这 5 个形状是同一个字符。



图 16-3 一个字符可以有很多不同的书写形式

在很多书面语体系中，根据一个字符在单词中位置的不同，同一个字符也会有不同的笔画形状。例如，图 16-4 中的 4 种笔画都表示字符 ARABIC LETTER AIN。[6] 图 16-4a 显示了 AIN 作为一个单独的字符时是如何书写的。图 16-4d 显示的是 AIN 在单词开头时的情形。图 16-4c 显示了 AIN 在单词中间的情形，而图 16-4b 显示的是 AIN 在单词结尾处的情形。[7]

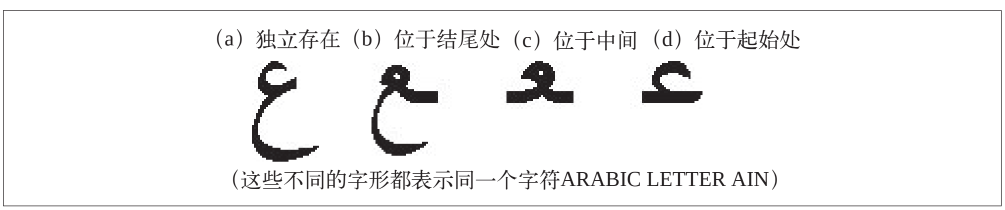


图 16-4 单个字符 ARABIC LETTER AIN 的 4 种与位置相关的书写形式

16.3.4 字形、连笔以及表示形式

不要把字符和字形混淆。字符是唯一的、抽象的语言“原子”。字形是画出每个字符时使用的特定方式。根据艺术形式和手法的不同，每个字符可以有很多不同的字形。[8]

注 6：AIN 的读音有些像 ayine 的发音，但发音朝向喉咙后部。

注 7：注意，阿拉伯语的单词是从右向左书写的。

注 8：很多人用术语“字形”来表示最终生成的位图图像，但从技术上说，字形是字符的内在形状，与字体和次要的艺术形式无关。进行这种区分不太容易，对我们的目的也没有什么用处。

同样，也不要把字符与表示形式混淆起来。为了让书法作品更好看，很多手写体和字体允许人们把相邻的字符漂亮地连写起来，称为连笔（ligatures），这样两个字符就平滑地连接在一起了。母语为英语的作者常把 F 和 I 结合为 FI 连笔（参见图 16-5a 和图 16-5b），而阿拉伯语的作者常把字符“LAM”和“ALIF”结合为一种很优雅的连笔（参见图 16-5c 和图 16-5d）。 378



图 16-5　连笔是相邻字符的另一种风格的表示形式，并非新的字符

这里给出一般的规则：如果用一种字形替代另一种的时候，文本的意思变了，那这些字形就是不同的字符。否则，它们就是同一个字符的不同风格的表示形式。[9]

16.3.5　编码后的字符集

根据 RFC 2277 和 2130 的定义，编码后的字符集把整数映射到字符。编码后的字符集经常用数组来实现 [10]，通过代码数值来索引（参见图 16-6）。数组的元素就是字符 [11]。

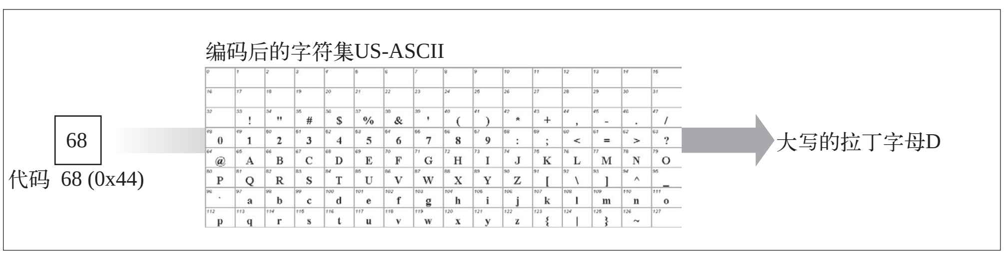


图 16-6　可以把编码后的字符集看作数组，把数值化的代码映射到字符

下面我们来看一些重要的编码后的字符集标准，包括具有历史意义的 US-ASCII 字符集、ASCII 的 iso-8859 扩展、日文的 JIS X 0201 字符集以及统一字符集（Universal Character Set，Unicode）。

注 9：语义和表示方式之间的区别并不总是很清晰的。为了实现的方便，已经为同一个字符的某些表示变体分配了不同的字符。不过我们还是尽量要避免这种做法。

注 10：数组可以是多维的，这样代码数字中的不同二进制码就可以索引到数组的不同维。

注 11：图 16-6 使用了一个网格来表示编码后的字符集。网格中的每个元素都包含一个字符图像。这些图像应看作是符号，图像“D”是字符 LATIN CAPITAL LETTER D 的简称，而不是任何特定的图形化字形。

1. US-ASCII:所有字符集的始祖

ASCII 是最著名的编码后字符集，早在 1968 年就由 ANSI 在标准 X3.4，“美国标准信
379 息交换代码”（American Standard Code for Information Interchange）中进行了标准化。ASCII 的代码值只是从 0 到 127，因此只需要 7 个二进制码就可以覆盖代码空间。ASCII 的推荐名称是 US-ASCII，这样可以和那些 7 位字符集的一些国际化变体区分开来。

HTTP 报文（首部、URI 等）使用的字符集是 US-ASCII。

2. iso-8859

iso-8859 字符集标准是 US-ASCII 的 8 位超集，使用二进制码的高位增加了一些国际化书面字符。由额外的二进制码提供的附加空间（多了 128 个代码）还不够大，甚至都不够所有的欧洲字符使用，更不用说亚洲字符了。因此 iso-8859 为不同地区定制了不同的字符集，如下所示。

iso-8859-1	西欧语言（例如，英语、法语）
iso-8859-2	中欧和东欧语言（例如，捷克、波兰）
iso-8859-3	南欧语言
iso-8859-4	北欧语言（例如，拉托维亚，立陶宛，格陵兰）
iso-8859-5	斯拉夫语（例如，保加利亚、俄罗斯、塞尔维亚）
iso-8859-6	阿拉伯语
iso-8859-7	希腊语
iso-8859-8	希伯来语
iso-8859-9	土耳其语
iso-8859-10	日耳曼和斯堪的纳维亚语言（例如，冰岛、因纽特）
iso-8859-15	对 iso-8859-1 的修改，包括了新的欧元字符

iso-8859-1 也称为 Latin1，是 HTML 的默认字符集。可以用它来表示大多数西欧语言的文本。因为 iso-8859-15 中包含了新的欧元符号，有过一些用它来代替 iso-8859-1 并作为 HTTP 默认编码后字符集的讨论。然而，由于 iso-8859-1 已经被广泛采用，要大范围地变更到 iso-8859-15 恐怕不是短时间内可以完成的。

3. JIS X 0201

JIS X 0201 是把 ASCII 扩展到日文半宽片假名字符的一个极小化的字符集。半宽片假名字符最早用在日文电报系统中。JIS X 0201 常常被称作 JIS Roman，JIS 是“Japanese Industrial Standard”（日文工业化标准）的缩写。

4. JIS X 0208与JIS X 0212

日文中包括数千个来自几个书面语系统中的字符。尽管可以（很痛苦地）勉强只使用 JIS X 0201 中的那 63 个基本的片假名字符，但实际使用中需要远比这个更完整
380 的字符集。

JIS X 0208 字符集是首个多字节日文字符集，它定义了 6879 个编码的字符，其中大多数是来源于中文的日本汉字。JIS X 0212 字符集又扩充了 6067 个字符。

5. UCS

UCS（Universal Character Set，统一字符集）是把全世界的所有字符整合到单一的编码后字符集的环球标准化成果。UCS 由 ISO 10646 定义。Unicode 是遵循 UCS 标准的商业化联合组织。UCS 具有能容纳百万以上字符的代码空间，不过基本集合只有大约 5 万个字符。

16.3.6　字符编码方案

字符编码方案规定如何把字符的代码数字打包装入内容比特，以及在另一端如何将其解包回字符代码（参见图 16-7）。字符编码方案有以下 3 种主要类型。

- 固定宽度

  固定宽度方式的编码用固定数量的比特表示每个编码后的字符。它们能被快速处理，但可能会浪费空间。

- 可变宽度（无模态）

  可变宽度方式的编码对不同的字符代码数字采用不同数量的比特。对于常用字符，这样可以减少需要的位数，而且还能在允许使用多字节来表示国际性字符的同时，保持对传统 8 位字符集的兼容性。

- 可变宽度（有模态）

  有模态的编码使用特殊的“转义”模式在不同的模态之间切换。例如，可以用有模态的编码在文本中使用多个互相有重叠的字符集。有模态的编码处理起来比较复杂，但它们可以有效地支持复杂的书写系统。

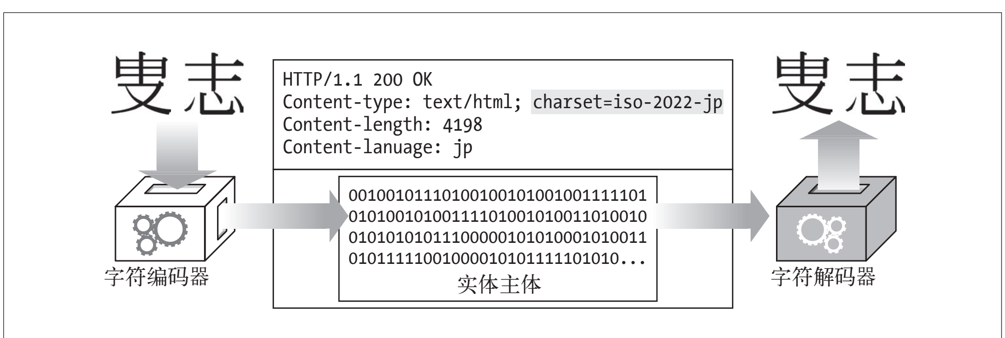


图 16-7　字符编码方案把字符代码编码为比特，并负责反向变换

381 下面我们来看一些常见的编码方案。

1. 8位

8 位固定宽度恒等编码把每个字符代码编码为相应的 8 位二进制值。它只能支持有 256 个字符代码范围的字符集。iso-8859 字符集家族系列使用的就是 8 位恒等编码。

2. UTF-8

UTF-8 是一种流行的为 UCS 设计的字符编码方案，UTF 表示 UCS 变换格式（UCS Transformation Format）。UTF-8 为字符代码值使用的是无模态的变宽编码，第一字节的高位表示编码后的字符所用的字节数，所需的每个后续字节都含有 6 位的代码值（参见表 16-2）。

如果编码后的第 1 字节的最高位是 0，长度就是 1 字节，剩余的 7 位就包含字符的代码。这样带来的美妙结果就是和 ASCII 兼容（但和 iso-8859 系列不兼容，因为 iso-8859 系列使用了最高位）。

表16-2　UTF-8 变宽无模态编码

字符代码的二进制位	字节1	字节2	字节3	字节4	字节5	字节 6
0-7	0ccccccc	-	-	-	-	-
8-11	110ccccc	10cccccc	-	-	-	-
12-16	1110cccc	10cccccc	10cccccc	-	-	-
17-21	11110ccc	10cccccc	10cccccc	10cccccc	-	-
22-26	111110cc	10cccccc	10cccccc	10cccccc	10cccccc	-
27-31	1111110c	10cccccc	10cccccc	10cccccc	10cccccc	10cccccc

例如，字符代码 90（ASCII 的“Z”）会被编码为 1 个字节（01011010），而代码 5073（13 位二进制值为 1001111010001）会被编码为 3 个字节：

1110000**1** 10**001111** 10**010001**

3. iso-2022-jp

iso-2022-jp 是互联网上的日文文档中广泛使用的编码。它是变宽、有模态的，所有值都不超过 128，以避免和不支持 8 位字符的软件出现兼容性问题。

编码上下文始终被设置为 4 种预设的字符集之一[12]，使用特殊的“转义序列”（escape sequence）在字符集之间切换。iso-2022-jp 的初始状态使用 US-ASCII 字符

注 12：iso-2022-jp 编码和这 4 种字符集是紧密绑定的，而其他一些编码是和特定的字符集无关的。

集，使用 3 个字节的转义序列可以切换到 JIS X 0201（JIS-Roman）字符集或大得多的 JIS X 0208-1978 和 JIS X 0208-1983 字符集。 382

表 16-3 中列出了这些转义序列。实际上，日文文本以 ESC $ @ 或 ESC $ B 开始，以 ESC (B 或 ESC (J 结束。

表16-3 iso-2022-jp的字符集切换转义序列

转义序列	转义后的字符集	每个代码的字节数
ESC (B	US-ASCII	1
ESC (J	JIS X 0201-1976 (JIS Roman)	1
ESC $ @	JIS X 0208-1978	2
ESC $ B	JIS X 0208-1983	2

在 US-ASCII 或 JIS-Roman 模态下，每个字符使用单个字节。当使用更大的 JIS X 0208 系列的字符集时，每个字符代码使用 2 个字节。该编码把发送的字节的值域范围限制在 33~126 之间[13]。

4. euc-jp

euc-jp 是另一种流行的日文编码。EUC 代表“Extended Unix Code”（扩展 Unix 代码），最早是为了在 Unix 操作系统上支持亚洲字符而开发的。

和 iso-2022-jp 类似，euc-jp 编码也是变长的，允许使用几种标准的日文字符集。但和 iso-2022-jp 不同的是，euc-jp 编码不是模态的。没有转义序列可以在不同模态之间切换。

euc-jp 支持 4 种编码后的字符集：JIS X 0201（JIS-Roman，对 ASCII 进行一些日文替换）、JIS X 0208、半宽片假名（最早在日文电报系统中使用的 63 个字符）以及 JIS X 0212。

编码 JIS Roman（它和 ASCII 兼容）的时候使用 1 个字节，对 JIS X 0208 和半宽片假名则使用 2 个字节，而对 JIS X 0212 使用 3 个字节。这种编码有点浪费空间但处理起来很简单。

表 16-4 概括了此编码的格局。

注 13：尽管每个字节只能有 94 个不同的值(33~126)，这也足够覆盖 JIS X 0208 系列字符集里面的所有字符了，因为这些字符集是按照 94 × 94 的网格来组织代码值的，所以 2 个字节足以覆盖 JIS X 0208 字符集中的全部字符代码。

表16-4　euc-jp编码值

哪个字节	编　码　值
JIS X 0201（有 94 个编码后的字符）	
第 1 个字节	33~126
JIS X 0208（有 6879 个编码后的字符）	
第 1 个字节	161~254
第 2 个字节	161~254
半宽片假名（有 63 个编码后的字符）	
第 1 个字节	142
第 2 个字节	161~223
JIS X 0212（有 6067 个编码后的字符）	
第 1 个字节	143
第 2 个字节	161~254
第 3 个字节	161~254

383

现在我们就介绍完了字符集和编码。下面一节将解释语言标记和 HTTP 如何使用语言标记把合适的内容传给受众。请参见附录 H，那里列出了详细的标准化字符集。

16.4　语言标记与HTTP

语言标记是命名口语的标准化字符串短语。

名字需要标准化，不然的话，有些人会把法语文档打上 French 标记，而有些其他人会用 Français，还有些人可能会用 France，更有些懒人可能会用 Fra 甚至是 F。标准化语言标记就可以避免这些混乱。

英语的标记是 en，德语的标记是 de，韩语的标记是 ko，等等。语言标记能够描述语言的地区变种和方言，比如巴西葡萄牙语的标记是 pt-BR、美式英语的标记是 en-US，汉语中的湖南话的标记是 zh-xiang。甚至还有个标准语言标记 i-klingon 是描述克林根语[14]的！

16.4.1　Content-Language首部

实体的 `Content-Language` 首部字段描述实体的目标受众语言。如果内容主要是给

注 14：科幻名著《星际迷航》（*Star Trek*）中的外星种族，加州大学伯克利分校的语言学博士马克乌克兰为其发明了一套语言。（译者注）

法语受众的，其 `Content-Language` 首部字段就将包含： 384

```
Content-Language: fr
```

`Content-Language` 首部不仅限于文本文档。音频片段、电影以及应用程序都有可能是面向特定语言受众的。任何面向特定语言受众的媒体类型都可以有 `Content-Language` 首部。在图 16-8 中，音频文件标记为面向纳瓦霍[15]（Navajo）听众。

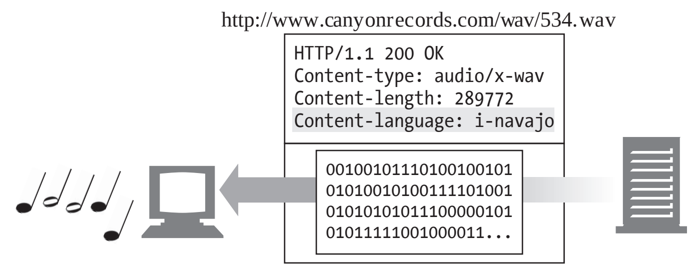


图 16-8　**Content-Language** 首部把名为 *Rain Song*（雨之歌）的音频片段标记为面向纳瓦霍听众

如果内容是面向多种语言受众的，可以列出多种语言。就像在 HTTP 规范中建议的，一份同时用英语和毛利语写的“Treaty of Waitangi”[16]（怀唐伊条约）译稿，可以这样描述：

```
Content-Language: mi, en
```

不过，不能仅根据有多种语言在实体中出现就认为它是面向多种语言受众的。为初学者编写的语言入门教材，比如“A First Lesson in Latin”（拉丁语第一课），显然是为英语受众准备的，应当只用 en 来描述。

16.4.2　Accept-Language首部

我们绝大多数人至少懂一种语言。HTTP 允许我们把语言方面的限制和优先选择都发送给网站服务器。如果网站服务器有以多种语言表示的资源版本，它就能把内容用我们最优选的语言表示出来。[17]

注 15：美国最大的印地安部落。（译者注）

注 16：英国政府与新西兰毛利酋长们在 1840 年签署的条约，是新西兰的“建国文献”。（译者注）

注 17：服务器也可以根据 `Accept-Language` 首部生成适合用户语言的动态内容，或据此选择图像，或选择适合目标语言的商业促销等。

这里有个例子，客户端请求西班牙语内容：

```
Accept-Language: es
```

可以在 `Accept-Language` 首部中放入多个语言标记以枚举所支持的全部语言及其优先顺序（从左到右）。这里有个例子，客户端首选英语，但也接受瑞士德语（标准语言标记是 de-CH）或其他德语变种（标记是 de）：

```
Accept-Language: en, de-CH, de
```

客户端使用 `Accept-Language` 首部和 `Accept-Charset` 首部请求可以理解的内容。第 17 章将研究这些机制的细节。

16.4.3　语言标记的类型

在 RFC 3066，“Tags for the Identification of Languages”（标识语言的标记）中记录了语言标记的标准化语法。可以用语言标记来表示：

- 一般的语言分类（比如 es 代表西班牙语）；
- 特定国家的语言（比如 en-GB 代表英国英语）；
385
- 语言的方言（比如 no-bok 指挪威的书面语）；
- 地区性的语言（比如 sgn-US-MA 代表美国马撒葡萄园岛上的手语）；
- 标准化的非变种语言（比如 i-navajo）；
- 非标准的语言（比如 x-snowboarder-slang[18]）。

16.4.4　子标记

语言标记有一个或多个部分，用连字号分隔，称为*子标记*：

- 第一个子标记称为*主子标记*，其值是标准化的；
- 第二个子标记是可选的，遵循它自己的命名标准；
- 其他尾随的子标记都是未注册的。

主子标记中只能含有字母（A ～ Z）。其后的子标记可以含有字母和数字，长度最多 8 个字符。图 16-9 中给出了一个示例。

注 18：描述由“酷爱滑雪者们”（shredders）使用的一种俚语。（译者注）

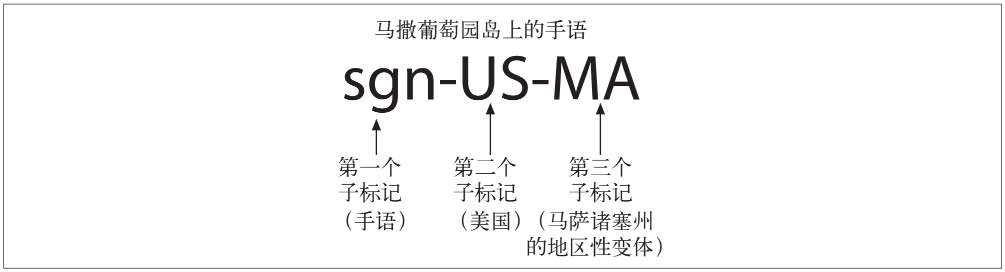


图 16-9　语言标记被分隔为子标记

16.4.5　大小写

所有的标记都是不区分大小写的，也就是说，标记 en 和 eN 是等价的。但是，习惯上用全小写来表示一般的语言，而用全大写来表示特定的国家。例如，fr 表示所有分类为法语的语言，而 FR 表示国家法国[19]。

16.4.6　IANA语言标记注册

第一个和第二个语言子标记的值由各种标准文档以及相关的维护组织定义。IANA[20] 依据 RFC 3066 中概括的规则来管理标准的语言标记列表。

如果语言标记由标准的国家和语言值组成，标记就不需要专门注册。只有那些无法
用标准的国家和语言值构成的语言标记才需要专门向 IANA 注册[21]。下面几节概括 386
了 RFC 3066 中关于第一子标记和第二子标记的标准。

16.4.7　第一个子标记——名字空间

第一个子标记通常是标准化的语言记号，选自 ISO 639 中的语言标准集合。不过也可以用字母 i 来标识在 IANA 中注册的名字，或用 x 表示私有的或者扩展的名字。下面是各种规则。

如果第一个子标记含有：

- 2 个字符，那就是来自 ISO 639[22] 和 639-1 标准的语言代码；

注 19：这种惯用法是 ISO 标准 3166 中推荐的。

注 20：参见 http://www.iana.org 和 RFC 2860。

注 21：编写本书的时候，IANA 中只显式注册了 21 种语言标记，包括汉语中的广东话（标记是 zh-yue）、新挪威语（标记是 no-nyn）、卢森堡语（标记是 i-lux）以及克林根语（语言标记是 i-klingon）。在因特网上使用的其余数百种口语的标记均由标准的组件构成。

注 22：参见 ISO 标准 639，“Codes for the representation of names of languages—Part2：Alpha-3code”（表示语言名字的代码——第 2 部分：Alpha-3 代码）。

- 3 个字符，那就是来自 ISO 639-2[23] 标准及其扩展的语言代码；
- 字母 i，该语言标记是在 IANA 显式注册的；
- 字母 x，该语言标记是私有的、非标准的，或扩展的子标记。

附录 G 中总结了 ISO 639 和 639-2 中的名字。表 16-5 中给出了一些示例。

表16-5　ISO 639和ISO 639-2中的语言代码示例

语　　言	ISO 639	ISO 639–2
阿拉伯语	ar	ara
汉语	zh	chi/zho
荷兰语	nl	dut/nla
英语	en	eng
法语	fr	fra/fre
德语	de	deu/ger
现代希腊语	el	ell/gre
希伯来语	he	heb
意大利语	it	ita
日语	ja	jpn
韩语	ko	kor
挪威语	no	nor
俄语	ru	rus
387 西班牙语	es	esl/spa
瑞典语	sv	sve/swe
土耳其语	tr	tur

16.4.8　第二个子标记——名字空间

第二个子标记通常是标准化的国家记号，选自 ISO 3166 中的国家代码和地区标准集合。不过也可以是在 IANA 注册过的其他字符串。下面是各种规则。

如果第二个子标记含有：

- 2 个字符，那就是 ISO 3166[24] 中定义的国家 / 地区；

注 23：参见 ISO 639-2，“Codes for the representation of names of languages—Part 2: Alpha-3 code”（表示语言名字的代码——第 2 部分：Alpha-3 代码）。

注 24：国家代码 AA、QM-QZ、XA-XZ 以及 ZZ 是在 ISO 3166 中保留的作为用户分配的代码。一定不能用这些值来构造语言标记。

- 3~8 个字符，可能是在 IANA 中注册的值；
- 单个字符，这是非法的情况。

表 16-6 中列出了 ISO 3166 中的部分国家代码。附录 G 中列出了完整的国家代码。

表16-6 ISO 3166中的国家代码示例

国　　家	代　　码
巴西	BR
加拿大	CA
中国	CN
法国	FR
德国	DE
梵蒂冈	VA
印度	IN
意大利	IT
日本	JP
黎巴嫩	LB
墨西哥	MX
巴基斯坦	PK
俄罗斯联邦	RU
英国	GB
美国	US

388

16.4.9 其余子标记——名字空间

除了最长可以到 8 个字符（字母和数字）之外，第三个和其后的子标记没有特殊规则。

16.4.10 配置和语言有关的首选项

可以在浏览器的配置文件中配置和语言有关的首选项。

网景公司的 Navigator 的设置方法是：编辑→首选项→语言，而微软公司的 Internet Explorer 浏览器的设置方法是：工具→ Internet 选项→语言。

16.4.11 语言标记参考表

为便于使用，附录 G 中给出了语言标记的参考表。

- 表 G-1 列出了在 IANA 注册的语言标记。
- 表 G-2 列出了 ISO 639 中的语言代码。
- 表 G-3 列出了 ISO 3166 中的国家代码。

16.5 国际化的URI

直到今天，URI 还没有为国际化提供足够的支持。除了少数（定义得很糟的）例外，URI 如今还是由 US-ASCII 字符的一个子集组成的。人们正在努力使主机名和 URL 的路径中能包含更丰富的集合中的字符，但直到现在，这些标准还没有被广泛接受和部署。现在让我们来回顾一下当前的一些尝试。

16.5.1 全球性的可转抄能力与有意义的字符的较量

URI 的设计者们希望世界上每个人都能通过电子邮件、电话、公告板，甚至无线电来共享 URI。他们还希望 URI 容易使用和记忆，但这两个目标是相互冲突的。

为了让世界各地的人们都能够便捷地输入、操控，以及共享 URI，设计者们为 URI 选择了常用字符的一个很有限的子集（基本的拉丁字母表中的字母、数字以及少数特殊符号）。世界上绝大多数软件和键盘都支持这个小的字符集合。

但不幸的是，限制了字符集的话，URI 就无法被全球的人们方便地使用和记忆。世
389 界上有很大一部分人甚至都不认识拉丁字母，他们几乎无法把 URI 当作抽象模式来记忆。

URI 的设计者们觉得确保资源标识符的*可转抄能力*（transcribability）和共享能力比让它们由最有意义的字符组成更加重要，因此（如今的）URI 基本上是由 ASCII 字符的受限子集构成的。

16.5.2 URI字符集合

URI 中允许出现的 US-ASCII 字符的子集，可以被分成*保留*、*未保留*以及*转义*字符这几类。未保留的字符可用于 URI 允许其出现的任何部分。保留的字符在很多 URI 中都有特殊的含义，因此一般来说不能使用它们。表 16-7 中列出了全部未保留、保留，以及转义字符。

表16-7 URI字符语法

<table>
<tr><th>字符类别</th><th>字符列表</th></tr>
<tr><td>未保留</td><td>[A-Za-z0-9] | "-" | "_" | "." | "!" | "~" | "*" | " ' " | "(" | ")"</td></tr>
<tr><td>保留</td><td>";" | "/" | "?" | ":" | "@" | "&" | "=" | "" | "$" | " , "</td></tr>
<tr><td>转义</td><td>"%" <HEX> <HEX></td></tr>
</table>

16.5.3　转义和反转义

URI 转义提供了一种安全的方式，可以在 URI 内部插入保留字符以及原本不支持的字符（比如各种空白）。每个转义是一组 3 字符序列，由百分号（%）后面跟上两个十六进制数字的字符。这两个十六进制数字就表示一个 US-ASCII 字符的代码。

例如，要在 URL 中插入一个空白（ASCII 32），可以用转义 %20，因为 20 是 32 的十六进制表示。类似地，如果想插入一个百分号并且不想让它被当作转义，就可以输入 %25，25 是百分号的 ASCII 代码的十六进制值。

图 16-10 展示了 URI 中的概念性字符是如何转换为当前字符集中字符的代码字节的。需要处理 URI 时，转义会被反转义回来，产生它们代表的 ASCII 代码的字节。

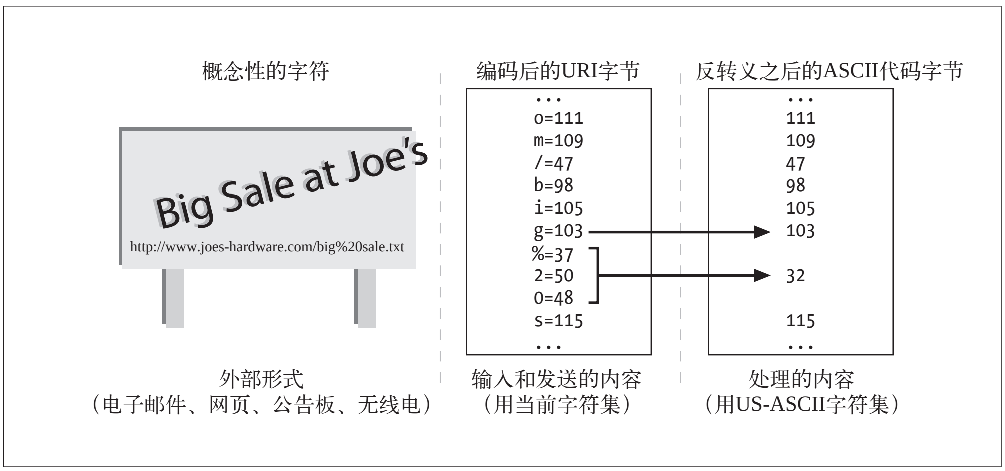


图 16-10　URI 中的字符在传输的时候要转义，但处理的时候要反转义

在内部处理时，HTTP 应用程序应当在传输和转发 URI 的时候保持转义不变。HTTP 应用程序应该仅在需要数据的时候才对 URI 进行转义。更重要的是，应用程序应该确保任何 URI 都不会被反转义 2 次，因为在转义的时候可能会把百分号编码进去，反转义出来之后，再转一次就会导致数据丢失。

16.5.4　转义国际化字符

需要注意的是，要转义的值本身应该在 US-ASCII 代码值的范围内（0 ～ 127）。某些应用程序试图用转义值来表示 iso-8859-1 中扩展的字符（代码范围在 128 ～ 255）。例如，网站服务器可能会错误地用转义来对包含了国际字符的文件名进行编 390
码。这样做是不对的，可能会使别的应用出问题。

例如，文件名 Sven Ölssen.html（包含了一个元音变音）可能被网站服务器编码为 Sven%20%D6lssen.html。把空格编码为 %20 是对的，但从技术上说，把 Ö 编码为 %D6 是非法的，因为代码 D6（十进制值 214）落在了 ASCII 代码范围之外。ASCII 只定义了最大值为 0x7F（十进制值 127）的代码。

16.5.5　URI中的模态切换

有些 URI 也用 ASCII 字符的序列来表示其他字符集中的字符。例如，可能使用 iso-2022-jp 编码插入“ESC (J”，切换到 JIS-Roman 字符集，用“ESC (B”切换回 ASCII 字符集。这在一些本地化的环境中可以工作，但这种方式没有进行良好的定义，而且没有标准化的方案来识别 URL 所使用的特定编码。正如 RFC 2396 的作者所说的那样：

> 不过，对于含有非ASCII字符的原始字符序列来说，境况更加复杂。如果可能用到多个字符集的话，传输表示字符序列的8位字节序列的因特网协议期待能有办法来识别所用的字符集[RFC 2277]。
>
> 然而，在通用的URI语法中没有提供进行这种识别的手段。个别的URI方案可以请求单一的字符集，定义默认的字符集，或提供指示所用字符集的方法。期待将来对这个规范的修改能为URI中的字符编码提供一种系统化的处理方案。

目前，URI 对国际化应用还不是非常友好。URI 的可移植性目标比语言灵活性方面的目标更重要。人们正在尽最大努力使 URI 更加国际化，但在短期内，HTTP 应用程序还是应当坚持使用 ASCII。它从 1968 年就出现了，所以只用它的话，一切还不
391 至于太糟。

16.6　其他需要考虑的地方

本节讨论在编写国际化的 HTTP 应用程序时，必须牢记的其他一些东西。

16.6.1　首部和不合规范的数据

HTTP 首部必须由 US-ASCII 字符集中的字符构成。不过，并不是所有的客户端和服务器都正确地实现了这一点，你可能会时不时收到一些代码值大于 127 的非法字符。

很多 HTTP 应用程序使用操作系统和库例程来处理字符（比如 Unix 中的字符分类库 ctype），但不是所有这些库都支持 ASCII 范围（0 ～ 127）之外的字符代码。

在某些情况下（一般来说，是较老的实现），当输入非 ASCII 字符时，这些库可能会返回不正确的结果，或者使应用程序崩溃。假设报文中含有非法数据，在使用这

些字符分类库来处理 HTTP 报文之前，要仔细阅读它们的文档。

16.6.2　日期

HTTP 的规范中明确定义了合法的 GMT 日期格式，但要知道并非所有 Web 服务器和客户端都遵守这些规则。例如，我们曾见过 Web 服务器发送的无效 HTTP Date（日期）首部中的月份是用本地语言表示的。

HTTP 应用程序应当尝试容忍一些不合规矩的日期，不能在接收的时候崩溃。不过也不是所有发送出来的日期都能被正确解释，如果日期无法解析，服务器应当谨慎处理。

16.6.3　域名

DNS 目前还不支持在域名中使用国际化的字符。现在正在进行支持多语言的域名的相关标准化工作，但还没有被广泛部署。

16.7　更多信息

万维网的极大成功意味着 HTTP 应用程序要继续在不同的语言和字符集之间交换更多的内容。更多关于多语言的多媒体这个重要但有些复杂的话题的信息，请参考下列资料来源。 392

16.7.1　附录

- 表 H-1 列出了在 IANA 注册的字符集标记。
- 表 G-1 列出了在 IANA 注册的语言标记。
- 表 G-2 列出了 ISO 639 中的语言代码。
- 表 G-3 列出了 ISO 3166 中的国家代码。

16.7.2　互联网的国际化

- http://www.w3.org/International/
  “Making the WWW Truly World Wide”（“使 WWW 真正遍布全球”）——W3C 国际化和本地化网站。

- http://www.ietf.org/rfc/rfc2396.txt
  RFC 2396，“Uniform Resource Identifiers（URI）：Generic Syntax”（“统一资源描述符：一般语法”），是 URI 的定义文档。该文档中包括了描述国际化 URI 中的字符集限制方面的章节。

- *CJKV Information Processing*（《中日韩越信息处理》）
  Ken Lunde 著，由 O'Reilly & Associates 公司出版。CJKV 是亚洲语言的电子化字符处理方面的权威经典。亚洲语言的字符集多种多样、复杂难懂，但这本书详细介绍了大型字符集的各种标准技术。

- http://www.ietf.org/rfc/rfc2277.txt
  RFC 2277，"IETF Policy on Character Sets and Languages"（"IETF 关于字符集和语言的策略"），其中记录了互联网工程指导组（Internet Engineering Steering Group，IESG）应用的当前策略，这些策略是为了配合互联网工程任务组（Internet Engineering Task Force，IETF）帮助各种互联网协议使用多种语言和字符交换数据而做的标准化努力。

16.7.3 国际标准

- http://www.iana.org/numbers.htm
  IANA 中有已注册的各种名字和数字编号的库。其中的 Protocol Numbers and Assignments Directory（协议编号和分配目录）中包含了因特网上使用的已注册字符集记录。因为有关国际间通信的大部分工作都是在 ISO 的领域内，而不是因特网的团体完成的，所以 IANA 的这份列表算不上详尽。

- http://www.ietf.org/rfc/rfc3066.txt
  393 RFC 3066，"Tags for the Identification of Languages"（"标识语言的标记"），描述了语言标记、它们的值，以及如何构造这些标记。

- "Codes for the representation of names of languages"（"表示语言名称的代码"）
  ISO 639:1988 (E/F)，国际标准化组织，第 1 版。

- "Codes for the representation of names of languages—Part 2: Alpha-3 code"（"表示语言名称的代码，第 2 部分：Alpha-3 代码"）
  ISO 639-2:1998，ISO TC46/SC4 和 TC37/SC2 联合工作组，第 1 版。

- "Codes for the representation of names of countries"（"表示国家名称的代码"）
  394 ISO 3166:1988 (E/F)，国际标准化组织，第 3 版。

第17章

内容协商与转码

一个 URL 常常需要代表若干不同的资源。例如那种需要以多种语言提供其内容的网站站点。如果某个站点（比如 Joe 的五金商店这样的站点）有说法语的和说英语的两种用户，它可能想用这两种语言提供网站站点信息。但在这种情况下，当用户请求 http://www.joes-hardware.com 时，服务器应当发送哪种版本呢？法文版还是英文版？

理想情况下，服务器应当向英语用户发送英文版，向法语用户发送法文版——用户只要访问 Joe 的五金商店的主页就可以得到相应语言的内容。幸运的是，HTTP 提供了*内容协商*方法，允许客户端和服务器作这样的决定。通过这些方法，单一的 URL 就可以代表不同的资源（比如，同一个网站页面的法语版和英语版）。这些不同的版本称为*变体*。

对于特定的 URL 来说，服务器还可以根据其他原则来决定发送什么内容给客户端最合适。在有些场合下，服务器甚至可以自动生成定制的页面。比如，服务器可以为手持设备把 HTML 页面转换成 WML 页面。这类动态内容变换被称为*转码*。这些变换动作是 HTTP 客户端和服务器之间进行内容协商的结果。

本章，我们将讨论内容协商和网站应用程序应该如何担负内容协商的责任。

17.1 内容协商技术

共有 3 种不同的方法可以决定服务器上哪个页面最适合客户端：让客户端来选择、服务器自动判定，或让中间代理来选。这 3 种技术分别称为客户端驱动的协商、服
395 务器驱动的协商以及透明协商（参见表 17-1）。本章，我们将研究每种技术的机制及其优缺点。

表17-1　内容协商技术概要

技　术	工作原理	优　点	缺　点
客户端驱动	客户端发起请求，服务器发送可选项的列表，客户端选择	在服务器端的实现最容易。客户端可以选择最合适的内容	增加了时延：为了获得正确的内容，至少要发送两次请求
服务器驱动	服务器检查客户端的请求首部集并决定提供哪个版本的页面	比客户端驱动的协商方式要快。HTTP 提供了 `q` 值机制，允许服务器近似匹配，还提供了 `Vary` 首部供服务器告知下游的设备如何对请求估值	如果结论不是很明确（比如首部集不匹配），服务器要做猜测
透明	某个中间设备（通常是缓存代理）代表客户端进行请求协商	免除了 Web 服务器的协商开销。比客户端驱动的协商要快	关于如何进行透明协商，还没有正式的规范

17.2 客户端驱动的协商

对于服务器来说，收到客户端请求时只是发回响应，在其中列出可用的页面，让客户端决定要看哪个，这是最容易的事情。很显然，这是服务器最容易实现的方式，而且客户端很可能选择到最佳的版本（只要列表中有让客户端选择的足够信息）。不利之处是每个页面都需要两次请求：第一次获取列表，第二次获取选择的副本。这种技术速度很慢且过程枯燥乏味，让用户厌烦。

从实现原理上来说，服务器实际上有两种方法为客户端提供选项：一是发送回一个 HTML 文档，里面有到该页面的各种版本的链接和每个版本的描述信息；另一种方法是发送回 HTTP/1.1 响应时，使用 300 Multiple Choices 响应代码。客户端浏览器收到这种响应时，在前一种情况下，会显示一个带有链接的页面；在后一种情况下，可能会弹出对话窗口，让用户做选择。不管怎么样，决定是由客户端的浏览器用户作出的。

除了增加时延并且对每个页面都要进行繁琐的多次请求之外，这种方法还有一个缺点：它需要多个 URL：公共页面要一个，其他每种特殊页面也都要一个。因此，比如说原始的请求地址是 www.joes-hardware.com，Joe 的服务器可能会回复某个页面，该页面里面有到 www.joes-hardware.com/english 和 www.joes-hardware.com/french 的链接。如果客户端想加书签的话，是要加在原始的公共页面上呢，还是加在选中的页面上呢？如果用户想把这个网站推荐给他的朋友，是告知 www.joes-hardware.com 这个地址好呢，还是只告诉他们讲英语的朋友 www.joes-hardware.com/english 这个地址？ 396

17.3 服务器驱动的协商

在前一节中，我们了解了客户端驱动的协商存在的若干缺点。大部分缺点都涉及客户端和服务器之间通信量的增长，这些通信量用来决定什么页面才是对请求的最佳响应。减少额外通信量的一种方法是让服务器来决定发送哪个页面回去，但为了做到这一点，客户端必须发送有关客户偏好的足够信息，以便服务器能够作出准确的决策。服务器通过客户端请求的首部集来获得这方面的信息。

有以下两种机制可供 HTTP 服务器评估发送什么响应给客户端比较合适。

- 检查内容协商首部集。服务器察看客户端发送的 `Accept` 首部集，设法用相应的响应首部与之匹配。
- 根据其他（非内容协商）首部进行变通。例如，服务器可以根据客户端发送的 `User-Agent` 首部来发送响应。

后面的小节将详细介绍这两种机制。

17.3.1　内容协商首部集

客户端可以用表 17-2 中列出的 HTTP 首部集发送用户的偏好信息。

表17-2　Accept首部集

首　　部	描　　述
Accept	告知服务器发送何种媒体类型
Accept-Language	告知服务器发送何种语言
Accept-Charset	告知服务器发送何种字符集
Accept-Encoding	告知服务器采用何种编码

注意，这些首部与第 15 章讨论的那些实体首部非常类似。不过，这两种首部的用途截然不同。正如第 15 章中所述，实体首部集像运输标签，它们描述了把报文从服务器传输给客户端的过程中必须的各种报文主体属性。而内容协商首部集是由客户端发送给服务器用于交换偏好信息的，以便服务器可以从文档的不同版本中选择出最符合客户端偏好的那个来提供服务。

397 服务器用表 17-3 中列出的实体首部集来匹配客户端的 Accept 首部集。

表17-3　Accept首部集和匹配的文档首部集

Accept首部	实体首部
Accept	Content-Type
Accept-Language	Content-Language
Accept-Charset	Content-Type
Accept-Encoding	Content-Encoding

注意，由于 HTTP 是无状态的协议（表示服务器不会在不同的请求之间追踪客户端的偏好），所以客户端必须在每个请求中都发送其偏好信息。

如果两个客户端都发送了 Accept-Language 首部，描述它们感兴趣的语言信息，服务器就能够决定发送 www.joes-hardware.com 的何种版本给哪个客户端了。让服务器自动选择发送回去的文档，减少了往返通信的时延，这种时延是客户端驱动模型中无法避免的。

然而，假设某个客户端偏好西班牙文，那服务器应当回送哪个版本的页面呢？英语还是法语？服务器只有两种选择：猜测或回退到客户端驱动模型，问客户端选择哪个。假如这个西班牙人碰巧懂一点英语，他可能会选择英文页面，这不是最理想的，

但它能解决问题。在这种情况下，这个西班牙人需要有办法传达更多与其偏好有关的信息，也就是他的确对英语略知一二，在没有西班牙语的时候，英语也行。

幸运的是，HTTP 提供了一种机制，可以让与这个西班牙人情况类似的客户端更详细地描述其偏好。这种机制就是*质量值*（简称 q 值）。

17.3.2 内容协商首部中的质量值

HTTP 协议中定义了质量值，允许客户端为每种偏好类别列出多种选项，并为每种偏好选项关联一个优先次序。例如，客户端可以发送下列形式的 `Accept-Language` 首部：

```
Accept-Language: en;q=0.5, fr;q=0.0, nl;q=1.0, tr;q=0.0
```

其中 q 值的范围从 0.0 ～ 1.0（0.0 是优先级最低的，而 1.0 是优先级最高的）。上面列出的那个首部，说明该客户端最愿意接收荷兰语（缩写为 nl）文档，但英语（缩写为 en）文档也行；无论如何，这个客户端都不愿意收到法语（缩写为 fr）或土耳其语（缩写为 tr）的版本。注意，偏好的排列顺序并不重要，只有与偏好相关的 q 值才是重要的。

服务器偶尔也会碰到找不到文档可以匹配客户端的任何偏好的情况。对于这种情况，服务器可以修改文档，也就是对文档进行转码，以匹配客户端的偏好。我们将在本章后面讨论这种机制。 398

17.3.3 随其他首部集而变化

服务器也可以根据其他客户端请求首部集来匹配响应，比如 `User-Agent` 首部。例如，服务器知道老版本的浏览器不支持 JavaScript 语言，这样就可以向其发送不含有 JavaScript 的页面版本。

在这种情况下，没有 q 值机制可供查找“最近似”的匹配。服务器或者去找完全匹配，或者简单地有什么就给什么，这取决于服务器的实现。

由于缓存需要尽力提供所缓存文档中正确的“最佳”版本，HTTP 协议定义了服务器在响应中发送的 `Vary` 首部。这个首部告知缓存（还有客户端和所有下游的代理）服务器根据哪些首部来决定发送响应的最佳版本。本章后面会更详细地讨论 `Vary` 首部。

17.3.4 Apache中的内容协商

这里概括了著名的 Web 服务器 Apache 是如何支持内容协商的。网站的内容提供者，

比如说 Joe，要负责为 Joe 的索引页面提供不同的版本。Joe 还必须把这些索引页面文件放在和站点相关的 Apache 服务器的适当目录下。用以下两种方式可以启用内容协商。

- 在网站目录中，为网站中每个有变体的 URI 创建一个 type-map（类型映射）文件。这个 type-map 文件列出了每个变体和其相关的内容协商首部集。
- 启用 MultiViews 指令，这样会使 Apache 自动为目录创建 type-map 文件。

1. 使用type-map文件

Apache 服务器需要知道 type-map 文件的命名规则。可以在服务器的配置文件中设置 handler 来说明 type-map 文件的后缀名。例如：

```
AddHandler type-map .var
```

这行就说明了后缀是 .var 的文件就是 type-map 文件。

399 这里给出一个 type-map 文件示例：

```
URI: joes-hardware.html

URI: joes-hardware.en.html
Content-type: text/html
Content-language: en
URI: joes-hardware.fr.de.html
Content-type: text/html;charset=iso-8859-2
Content-language: fr, de
```

根据这个 type-map 文件，Apache 服务器就知道要发送 joes-hardware.en.html 给请求英语版的客户端，发送 joes-hardware.fr.de.html 给请求法语版的客户端。Apache 服务器也支持质量值，具体信息请参阅它的文档。

2. 使用MultiView

为了使用 MultiView，必须在网站目录下的 access.conf 文件中的适当小节（<*Directory*>、<*Location*>，或 <*Files*>）使用 OPTION 指令来启用它。

如果启用了 MultiView，而浏览器又请求了名为 joes-hardware 的资源，服务器就会查找所有名字中含有 joes-hardware 的文件，并为它们创建 type-map 文件。服务器会根据名字猜测其对应的内容协商首部集。例如，法语版的 joes-hardware 应当含有 *.fr*。

17.3.5 服务器端扩展

另一种在服务器端实现内容协商的方法是使用服务器端扩展，比如微软的动态服务

器页面（Microsoft's Active Server Pages，ASP）。参见第 8 章中关于服务器端扩展的综述。

17.4 透明协商

透明协商机制试图从服务器上去除服务器驱动协商所需的负载，并用中间代理来代表客户端以使与客户端的报文交换最小化。假定代理了解客户端的预期，这样就可以代表客户端与服务器协商（在客户端请求内容的时候，代理已经收到了客户端的预期）。为了支持透明内容协商，服务器必须有能力告知代理，服务器需要检查哪些请求首部，以便对客户端的请求进行最佳匹配。HTTP/1.1 规范中没有定义任何透明协商机制，但定义了 `Vary` 首部。服务器在响应中发送了 `Vary` 首部，以告知中间节点需要使用哪些请求首部进行内容协商。

代理缓存可以为通过单个 URL 访问的文档保存不同的副本。如果服务器把它们的决策过程传给缓存，这些代理就能代表服务器与客户端进行协商。缓存同时也是进行内容转码的好地方，因为部署在缓存里的通用转码器能对任意服务器，而不仅仅是一台服务器传来的内容进行转码。图 17-3 中展示了缓存对内容进行转码的情况，本章后面会更详细地探讨。 400

17.4.1 进行缓存与备用候选

对内容进行缓存的时候是假设内容以后还可以重用。然而，为了确保对客户端请求回送的是正确的已缓存响应，缓存必须应用服务器在回送响应时所用到的大部分决策逻辑。

前一节描述了客户端发送的 `Accept` 首部集，以及为了给每条请求选择最佳的响应，服务器使用的与这些首部集匹配的相应实体首部集。缓存也必须使用相同的首部集来决定回送哪个已缓存的响应。

图 17-1 展示了涉及缓存的正确及错误的操作序列。缓存把第一个请求转发给服务器，并存储其响应。对于第二个请求，缓存根据 URL 查找到了匹配的文档。但是，这份文档是法语版的，而请求者想要的是西班牙语版的。如果缓存只是把文档的法语版本发给请求者的话，它就犯了错误。

因此，缓存也应该把第二条请求转发给服务器，并保存该 URL 的响应与“备用候选”响应。缓存现在就保存了同一个 URL 的两份不同的文档，与服务器上一样。这 401
些不同的版本称为*变体*（variant）或*备用候选*（alternate）。内容协商可看成是为客户端请求选择最合适变体的过程。

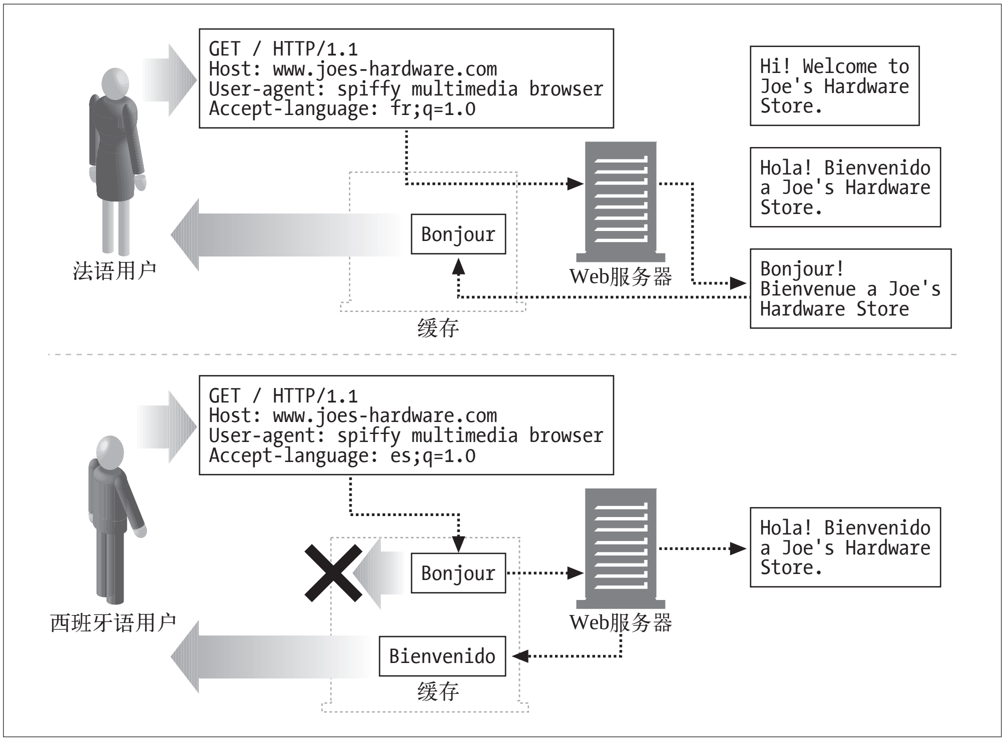


图 17-1　缓存根据内容协商首部发送给客户端正确的响应

17.4.2　Vary首部

这里是浏览器和服务器发送的一些典型的请求及响应首部：

```
GET http://www.joes-hardware.com/ HTTP/1.0
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.73 [en] (WinNT; U)
Host: www.joes-hardware.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/
    png, */*
Accept-Encoding: gzip
Accept-Language: en, pdf
Accept-Charset: iso-8859-1, *, utf-8

HTTP/1.1 200 OK
Date: Sun, 10 Dec 2000 22:13:40 GMT
Server: Apache/1.3.12 OpenSSL/0.9.5a (Unix) FrontPage/4.0.4.3
Last-Modified: Fri, 05 May 2000 04:42:52 GMT
Etag: "1b7ddf-48-3912514c"
Accept-Ranges: Bytes
Content-Length: 72
Connection: close
Content-Type: text/html
```

然而，如果服务器的决策不是依据 Accept 首部集，而是比如 User-Agent 首部的话，情况会如何？这不像听起来这么极端。例如，服务器可能知道老版本的浏览器不支持 JavaScript 语言，因此可能会回送不包含 JavaScript 的页面版本。如果服务器是根据其他首部来决定发送哪个页面的话，缓存必须知道这些首部是什么，这样才能在选择回送的页面时做出同样的逻辑判断。

HTTP 的 Vary 响应首部中列出了所有客户端请求首部，服务器可用这些首部来选择文档或产生定制的内容（在常规的内容协商首部集之外的内容）。例如，若所提供的文档取决于 User-Agent 首部，Vary 首部就必须包含 User-Agent。

当新的请求到达时，缓存会根据内容协商首部集来寻找最佳匹配。但在把文档提供给客户端之前，它必须检查服务器有没有在已缓存响应中发送 Vary 首部。如果有 Vary 首部，那么新请求中那些首部的值必须与旧的已缓存请求里相应的首部相同。因为服务器可能会根据客户端请求的首部来改变响应，为了实现透明协商，缓存必须为每个已缓存变体保存客户端请求首部和相应的服务器响应首部，参见图 17-2。 402

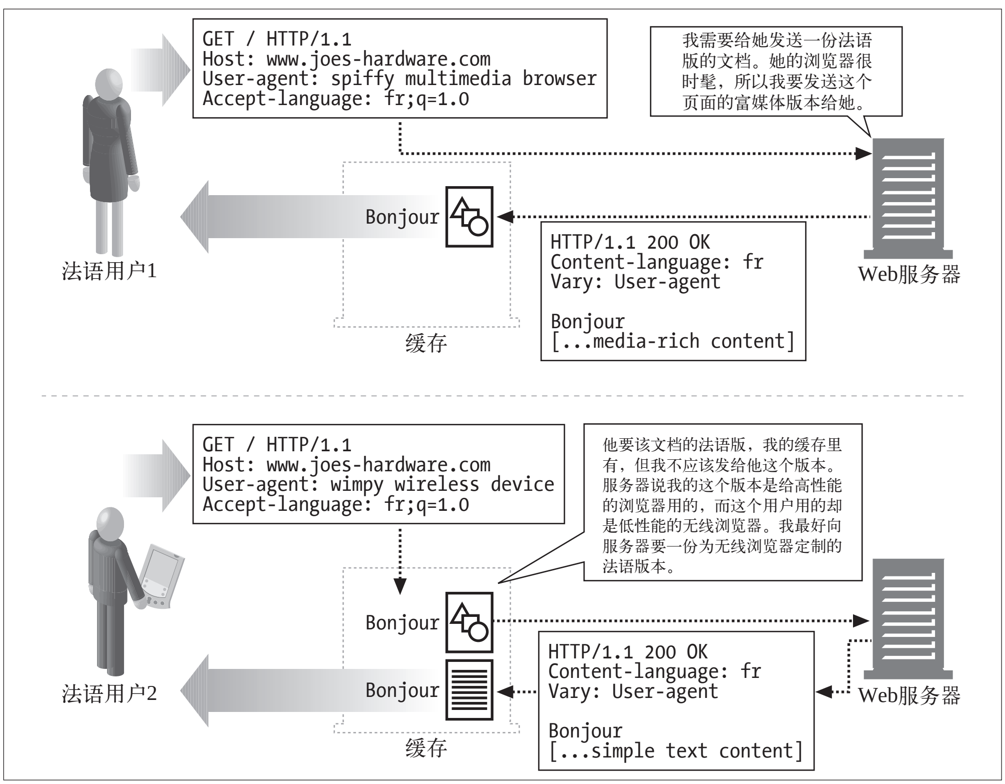


图 17-2　如果服务器根据特定的请求首部集来选择变体，缓存必须在发送回缓存的响应之前，检查常规的内容协商首部集和这些请求首部

如果某服务器的 Vary 首部看起来像下面这样，大量不同的 User-Agent 和 Cookie 值将会产生非常多的变体：

```
Vary: User-Agent, Cookie
```

缓存必须为每个变体保存其相应的文档版本。当缓存执行查找时，首先会对内容协商首部集进行内容匹配，然后比较请求的变体与缓存的变体。如果无法匹配，缓存就从原始服务器获取文档。

17.5 转码

我们已经讨论了一个机制，该机制可以让客户端和服务器从某个 URL 的一系列文档
403 中挑选出最适合客户端的文档。实现这些机制的前提是，存在一些满足客户端需求的文档——不管是完全满足还是在一定程度上满足。

然而，如果服务器没有能满足客户端需求的文档会怎么样呢？服务器可以给出一个错误响应。但理论上，服务器可以把现存的文档转换成某种客户端可用的文档。这种选项称为*转码*。

表 17-4 列出了一些假设的转码。

表17-4　假设的转码

转换之前	转换之后
HTML 文档	WML 文档
高分辨率图像	低分辨率图像
彩色图像	黑白图像
有多个框架的复杂页面	没有很多框架或图像的简单文本页面
有 Java 小应用程序的 HTML 页面	没有 Java 小应用程序的 HTML 页面
有广告的页面	去除广告的页面

有 3 种类别的转码：格式转换、信息综合以及内容注入。

17.5.1 格式转换

*格式转换*是指将数据从一种格式转换成另一种格式，使之可以被客户端查看。通过 HTML 到 WML 的转换，无线设备就可以访问通常供桌面客户端查看的文档了。通过慢速连接访问 Web 页面的客户端并不需要接收高分辨率图像，如果通过格式转换降低图像分辨率和颜色来减小图像文件大小的话，这类客户端就能更容易地查看图像比较丰富的页面了。

格式转换可以由表 17-2 中列出的内容协商首部集来驱动，但也能由 `User-Agent` 首部来驱动。注意，内容转换或转码与内容编码或传输编码是不同的，后两者一般用于更高效或安全地传输内容，而前两者则可使访问设备能够查看内容。

17.5.2 信息综合

从文档中提取关键的信息片段称为*信息综合*（information synthesis），这是一种有用的转码操作。这种操作的例子包括根据小节标题生成文档的大纲，或者从页面中删除广告和商标。 404

根据内容中的关键字对页面分类是更精细的技术，有助于总结文档的精髓。这种技术常用于 Web 页面分类系统中，比如门户网站的 Web 页面目录。

17.5.3 内容注入

前面描述的两类转码通常会减少 Web 文档的内容，但还有另一类转换会增加文档的内容，即*内容注入*转码。内容注入转码的例子有自动广告生成器和用户追踪系统。

设想一下，一个能往途经的每个 HTML 页面中自动添加广告的广告植入转码器是多么的诱人（当然也很烦人）。这类转码操作只能动态进行——它必须即时添加与当前的特定用户有关，或针对特定用户的广告。也可以构建用户追踪系统，在页面中动态增加内容，用于收集用户查看页面和客户端浏览方式的统计信息。

17.5.4 转码与静态预生成的对比

转码的替代做法是在 Web 服务器上建立 Web 页面的不同副本，例如一个是 HTML，一个是 WML；一个图像分辨率高，一个图像分辨率低；一个有多媒体内容，一个没有。但是，这种方法不是很切合实际，原因很多：某个页面中的任何小改动都会牵扯很多页面，需要很多空间来存储各页面的不同版本，而且使页面编目和 Web 服务器编程（以提供正确的版本）变得更加困难。有些转码操作，比如广告插入（尤其是定向广告插入），就不能静态实现——因为插入什么广告和请求页面的用户有关。

对单一的根页面进行即时转换，是比静态的预生成更容易的解决方案。但这样会在提供内容时增加时延。不过有时候其中一些计算可以由第三方进行，这样就减少了 Web 服务器上的计算负荷——比如可以由代理或缓存中的外部 Agent 完成转换。图 17-3 显示了在代理缓存中进行的转码。

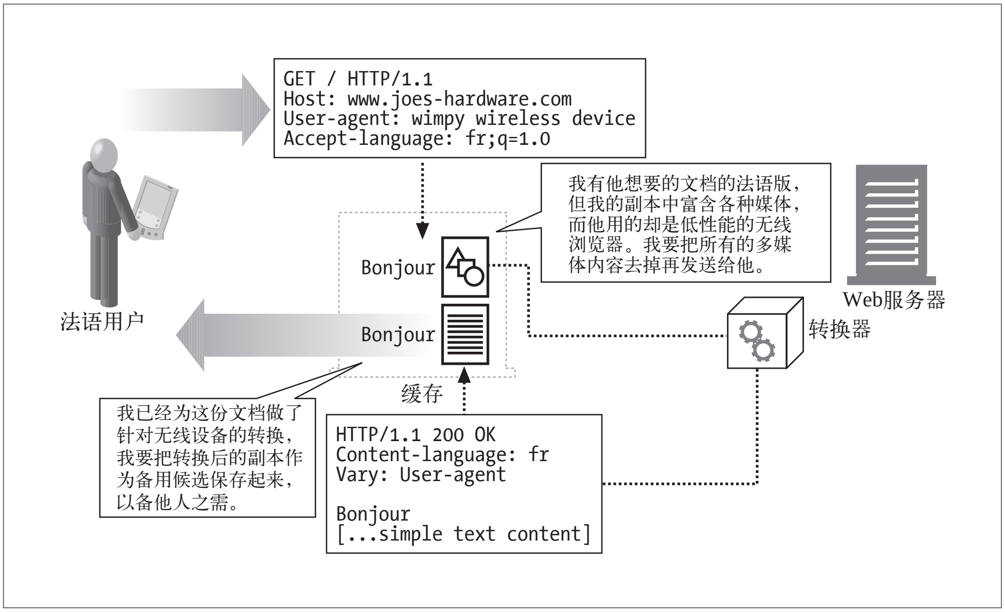


图 17-3　在代理缓存中进行转换或转码

17.6　下一步计划

由于以下两个原因，内容协商这个话题不只限于 Accept 和 Content 这两个首部集。

- HTTP 中的内容协商受到一些性能方面的限制。在各种变体中搜索合适的内容，或尽力“猜测”最佳匹配，都会有很大开销。有没有什么办法能专注内容协商协
405 议以使这个过程更高效？ RFC 2295 和 RFC2296 尝试着对这个问题进行了研究，以提供透明的 HTTP 内容协商。
- HTTP 不是唯一需要进行内容协商的协议。在其他一些情况下，客户端也需要和服务器交互以便获得对客户端请求来说最好的答案，流媒体和传真就是另外两个例子。能否在 TCP/IP 应用层协议之上开发出通用的内容协商协议呢？内容协商工作组（Content Negotiation Working Group）就是专门为这个问题而成立的。这个工作组目前已经停止工作了，不过它提出了若干个 RFC。在下一节中，我们给出了这个组的网站链接。

17.7　更多信息

从下面的因特网草案和在线文档中可以获得更多内容协商方面的信息。

- http://www.ietf.org/rfc/rfc2616.txt
  RFC 2616，“Hypertext Transfer Protocol–HTTP/1.1”（“超文本传输协议 HTTP/1.1”），这是 HTTP 协议的当前版本，也是 HTTP/1.1 的官方规范。这份规范行文流畅、组织良好，是份详实的 HTTP 参考文献。不过对那些希望学习 HTTP 背后的各种概念和决策动机、弄清理论与实践不同之处的读者来说，它就不是很理想了。我们希望本书能补足背后的这些概念，使读者能更好地利用这份规范。 406

- http://www.ietf.org/rfc/rfc2295.txt
  RFC 2295，“Transparent Content Negotiation in HTTP”（“HTTP 中的透明内容协商”），这是一份备忘录，描述了建立在 HTTP 之上的透明内容协商协议。这份备忘录目前还是实验性的。

- http://www.ietf.org/rfc/rfc2296.txt
  RFC 2296，“HTTP Remote Variant Selection Algorithm RVSA 1.0”（“HTTP 远程变体选择算法 RVSA1.0”），这份备忘录描述了为特定的 HTTP 请求透明地选择“最佳”内容的算法。这份备忘录目前还是实验性的。

- http://www.ietf.org/rfc/rfc2936.txt
  RFC 2936，“HTTP MIME Type Handler Detection”（“HTTP MIME 类型处理器检测”），这份备忘录描述了一种用来判定浏览器支持的 MIME 类型处理器的方法。如果 Accept 首部不够明确的话，这种方法就能派上用场。

- http://www.imc.org/ietf-medfree/index.htm
  这个链接指向内容协商（简称 CONNEG）工作组网站。该工作组专注于 HTTP、传真和打印方面的透明内容协商。这个工作组目前已停止工作。 407

第五部分

内容发布与分发

第五部分讲述了 Web 内容发布和传播的各种技术。

- 第 18 章介绍了在现代的 Web 托管环境中部署服务器的若干方法，HTTP 对虚拟 Web 托管的支持以及如何在地理上相距遥远的服务器之间复制内容。
- 第 19 章讨论了创建 Web 内容并将其放置到 Web 服务器上去的各种技术。
- 第 20 章探讨了各种将来访的 Web 流量分发到一组服务器上的技术和工具。 408～410
- 第 21 章解释了日志的各种格式和各种常见问题。

第18章

Web主机托管

当你把资源放在公共的 Web 服务器上时，因特网社区就可以使用它们了。这些资源可以是简单的文本文件或图像，也可以是复杂的实时导航地图或电子商务购物网关。能够将这些由不同组织拥有的种类繁多的资源便利地发布到网站上，并将其放置在能以合理价格提供很好性能的 Web 服务器上，是很关键的。

对内容资源的存储、协调以及管理的职责统称为 Web *主机托管*。主机托管是 Web 服务器的主要功能之一。保存并提供内容，记录对内容的访问以及管理内容都离不开服务器。如果不想自行管理服务器所需的软硬件，就需要主机托管服务，即*托管者*。托管者出租服务和网站管理维护业务，并提供各种不同程度的安全级别、报告及易用性。托管者通常把很多网站放在一些强大的 Web 服务器上联合运行，这样可以获得更高的成本效益、可靠性和性能。

本章讲解 Web 主机托管服务中的某些重要特征和它们如何与 HTTP 应用程序交互。本章的主要内容包括：

- 不同的网站如何被“虚拟地托管”在同一个服务器上，这样会对 HTTP 产生怎样的影响；
- 在很大的流量压力下，如何确保网站更可靠；
- 如何使网站加载更快。

18.1 主机托管服务

在万维网的早期，每个组织自行购买自己的计算机硬件，搭建自己的计算机房，申请自己的网络连接，并管理自己的 Web 服务器软件。

随着 Web 迅速成为主流，每人都想要一个网站，但很少有人有能力或时间来搭建带
411 空调的服务器机房，注册域名，或购买网络带宽。为了满足人们的迫切需求，出现了很多新的企业，提供了专业化管理的 Web 主机托管服务。服务级别有多种，从物理上的设备管理（提供空间、空调以及线缆）到完整的 Web 主机托管，顾客只需要提供内容就行了。

本章主要探讨托管 Web 服务器要提供什么服务。网站运作需要的很多东西（例如，它支持不同语言的能力和进行安全的电子商务交易的能力）都取决于托管 Web 服务器提供的功能。

简单例子——专用托管

假设 Joe 的五金商店和 Mary 的古董拍卖店都需要大容量的网站。Irene 网络服务提供商那里有很多机架，机架上全是一样的高性能 Web 服务器，可以租给 Joe 和 Mary，

这样，他俩就不用自行购买自己的服务器并管理服务器软件了。

在图 18-1 中，Joe 和 Mary 都签约使用 Irene 的网络服务提供商提供的专用 Web 托管服务。Joe 租了专用的 Web 服务器，该服务器是 Irene 网络服务提供商购买和维护的。Mary 也从 Irene 网络服务提供商那里租了另一个专用服务器。Irene 网络服务提供商大批量地购买服务器硬件，它们选择的硬件经久耐用且相对便宜。如果 Joe 或 Mary 的网站变得更受欢迎，Irene 网络服务提供商可以立刻给 Joe 或 Mary 提供更多的服务器。

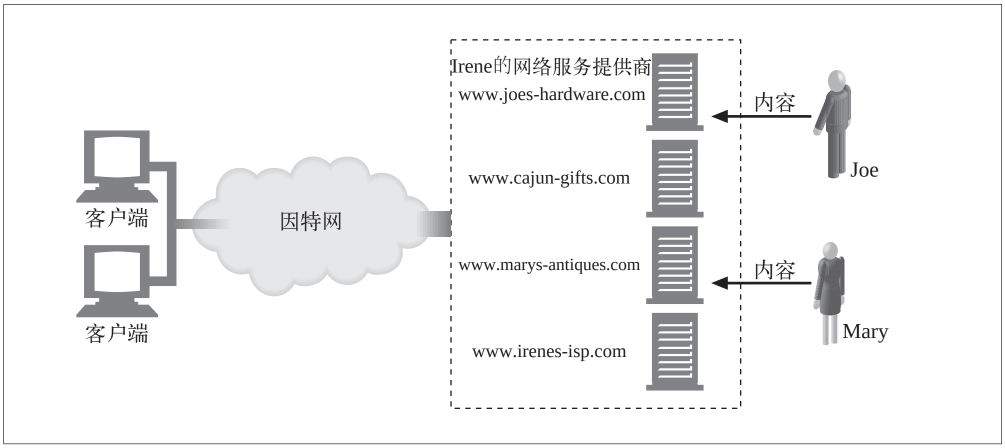


图 18-1　外包的专用托管服务

在这个例子中，浏览器向 Joe 服务器的 IP 地址发送对 www.joes-hardware.com 的 HTTP 请求，向 Mary 服务器（不同于 Joe）的 IP 地址发送对 www.marys-antiques.com 的请求。 412

18.2　虚拟主机托管

许多人想要在 Web 上展现自己，但他们的网站流量都不大。对这些人来说，使用专用的 Web 服务器可能有点儿大材小用，因为他们每月花费数百美元租来的服务器大部分时间都是空闲的！

许多 Web 托管者通过让一些顾客共享一台计算机来提供便宜的 Web 主机托管服务。这称为*共享主机托管*或*虚拟主机托管*。每个网站看起来是托管在不同的服务器上，但实际上是托管在同一个物理服务器上。从最终用户的角度来看，被虚拟托管的网站应当和托管在专用服务器上的网站没什么区别。

从成本效益、空间以及管理方面考虑，提供虚拟主机托管的公司希望能在同一个

服务器上托管数十、上百，甚至上千个网站——但这不一定意味着上千个网站是用一台 PC 机来提供服务的。托管者可以创建成排同样的服务器，称为服务器集群（server farm），把负载分摊在群里的服务器上。因为群里的每台服务器都一样，并且托管了许多虚拟网站，所以管理起来更加方便。（我们将在第 20 章更详细地介绍服务器集群。）

当 Joe 和 Mary 刚开始商务运作时，他们可能会选择虚拟主机托管，以节省费用，直到他们网站的流量规模达到值得使用专用服务器的水平为止（参见图 18-2）。

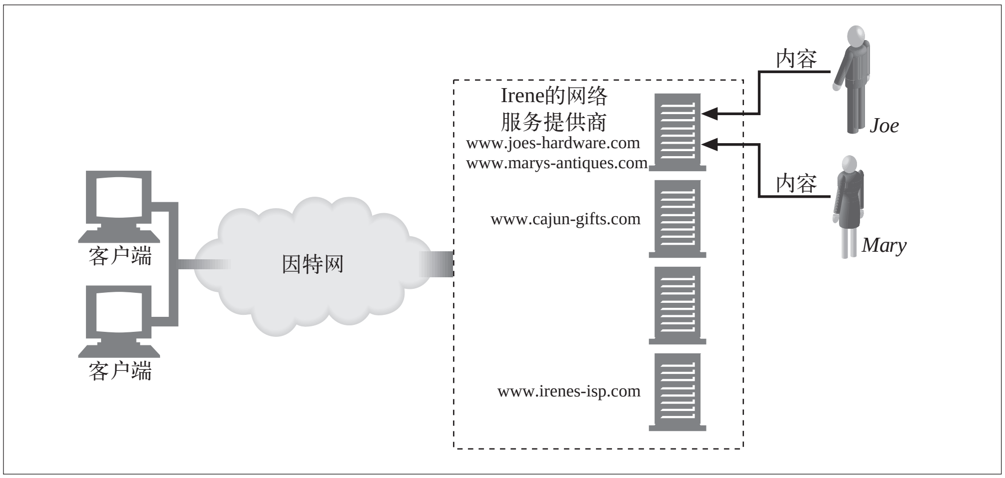


图 18-2　外包的虚拟主机托管

18.2.1　虚拟服务器请求缺乏主机信息

不幸的是，HTTP/1.0 中的一个设计缺陷会使虚拟主机托管者抓狂。HTTP/1.0 规范中
413 没有为共享的 Web 服务器提供任何方法来识别要访问的是所托管的哪个虚拟网站。

回想一下，HTTP/1.0 请求在报文中只发送了 URL 的路径部分。如果要访问 http://www.joes-hardware.com/index.html，浏览器会连接到服务器 www.joes-hardware.com，但 HTTP/1.0 请求中只提到 GET /index.html，没有提到主机名。如果服务器虚拟托管了多个站点，就没有足够的信息能指出要访问的是哪个虚拟网站。图 18-3 就是这样的一个示例。

- 如果客户端 A 试图访问 http://www.joes-hardware.com/index.html，请求 GET /index.html 将被发送到共享的 Web 服务器。
- 如果客户端 B 试图访问 http://www.marys-antiques.com/index.html，同样的请求 GET /index.html 也将被发送到共享的 Web 服务器。

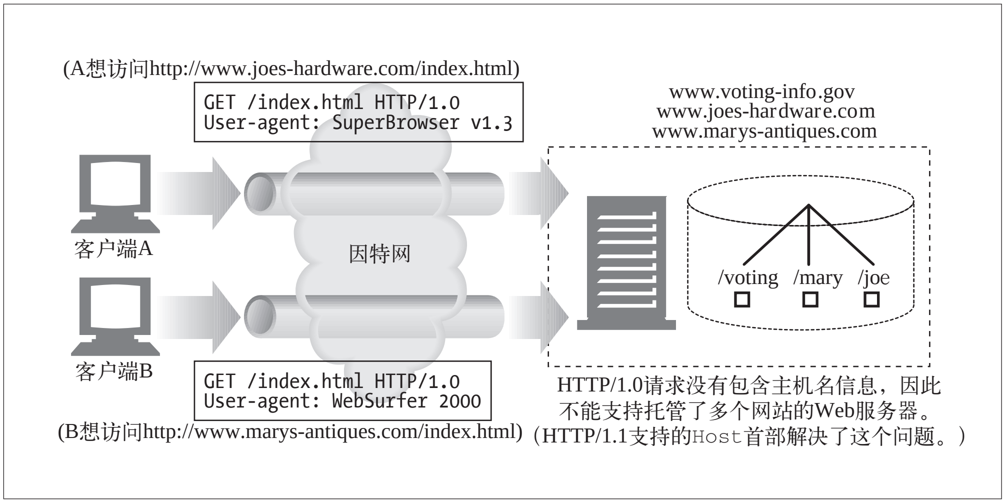


图 18-3　HTTP/1.0 服务器请求中没有主机名信息

就 Web 服务器而言，没有足够的信息可供其判断究竟要访问的是哪个网站。尽管请求的是完全不同的文档（来自不同的网站），但这两个请求看起来是一样的，这是因为网站的主机信息已经从请求中剥离了。

我们已经在第 6 章中介绍过，HTTP 替代物（反向代理）和拦截代理也都需要明确的站点信息。

18.2.2　设法让虚拟主机托管正常工作

缺失的主机信息是原始 HTTP 规范的疏忽，它错误地假设了每个 Web 服务器上只托管了一个网站。HTTP 的设计者没有为进行虚拟主机托管的共享服务器提供支持。正因为如此，URL 中的主机名信息被当作冗余信息剥离了，只要求发送路径部分。

因为早期的规范没有考虑到虚拟主机托管，Web 托管者需要开发变通的方案和约定来支持共享的虚拟主机托管。这个问题本可以通过要求所有 HTTP 请求报文发送完整的 URL 而不只是路径部分来简单地解决。而 HTTP/1.1 的确要求服务器能够处理 HTTP 报文请求行上的完整 URL，但将现存的应用程序都升级到这个规范还需要很长时间。在此期间，涌现了以下 4 种技术。 414

- 通过 URL 路径进行虚拟主机托管
  在 URL 中增添专门的路径部分，以便服务器判断是哪个网站。

- 通过端口号进行主机托管
  为每个站点分配不同的端口号，这样请求就由 Web 服务器的单独实例来处理。

- 通过 IP 地址进行主机托管
  为不同的虚拟站点分配专门的 IP 地址，把这些地址都绑定到一台单独的机器上。这样，Web 服务器就可以通过 IP 地址来识别网站名了。

- 通过 `Host` 首部进行主机托管
  很多 Web 托管者向 HTTP 的设计者施压，要求解决这个问题。HTTP/1.0 的增强版和 HTTP/1.1 的正式版定义了 `Host` 请求首部来携带网站名称。Web 服务器可以通过 `Host` 首部识别虚拟站点。

接下来详细介绍每种技术。

1. 通过URL路径进行虚拟主机托管

可以通过分配不同的 URL 路径，用这种笨方法把共享服务器上的虚拟站点隔离开。例如，可以给每个逻辑网站一个专门的路径前缀。

- Joe 的五金商店可以是：http://www.joes-hardware.com/joe/index.html。
- Mary 的古董拍卖店可以是：http://www.marys-antiques.com/mary/index.html。

当请求到达服务器时，其中并没有主机名信息，但服务器可以通过路径来区分它们。

- 请求 Joe 的五金商店的网址是 GET /joe/index.html。
- 请求 Mary 的古董拍卖店的网址是 GET /mary/index.html。

这不是一个好办法。/joe 和 /mary 这样的前缀是多余的（主机名中已经提到 joe 了）。更糟的是，描述主页链接的常见约定：http://www.joes-hardware.com 或 http://www.joes-hardware.com/index.html 都不能用了。

总之，按 URL 来进行虚拟主机托管是一个糟糕的解决方案，很少会用到。

2. 通过端口号进行虚拟主机托管

除了修改路径名，还可以在 Web 服务器上为 Joe 和 Mary 的网站分配不同的端口号。
415 不再使用端口 80，而是采用其他端口号，例如，Joe 用 82 Mary，用 83。但这个解决方案也有同样的问题：终端用户不会乐意在 URL 中指定非标准的端口号。

3. 通过IP地址进行虚拟主机托管

一个更常用的、更好的方法是通过 IP 地址进行虚拟化。每个虚拟网站都分配一个或多个唯一的 IP 地址。所有虚拟网站的 IP 地址都绑定到同一个共享的服务器上。服务器可以查询 HTTP 连接的目的 IP 地址，并以此来判断客户端的目标网站。

比方说，托管者把 IP 地址 209.172.34.3 分配给 www.joes-hardware.com，把 IP 地址 209.172.34.4 分配给 www.marys-antiques.com，把这两个 IP 地址都绑定到同一个物理服务器上。Web 服务器就能使用目的 IP 地址来识别用户请求的是哪个虚拟站点了，参见图 18-4。

- 客户端 A 获取 http://www.joes-hardware.com/index.html。
- 客户端 A 查询 www.joes-hardware.com 的 IP 地址，得到 209.172.34.3。
- 客户端 A 打开到共享服务器的 TCP 连接，目的地址是 209.172.34.3。
- 客户端 A 发送请求，内容为 GET /index.html HTTP/1.0。
- 在 Web 服务器提供响应之前，它注意到实际的目的 IP 地址（209.172.34.3），判断出这是 Joe 的五金网站的虚拟 IP 地址，就根据子目录 /joe 来完成请求。返回的是文件 /joe/index.html。

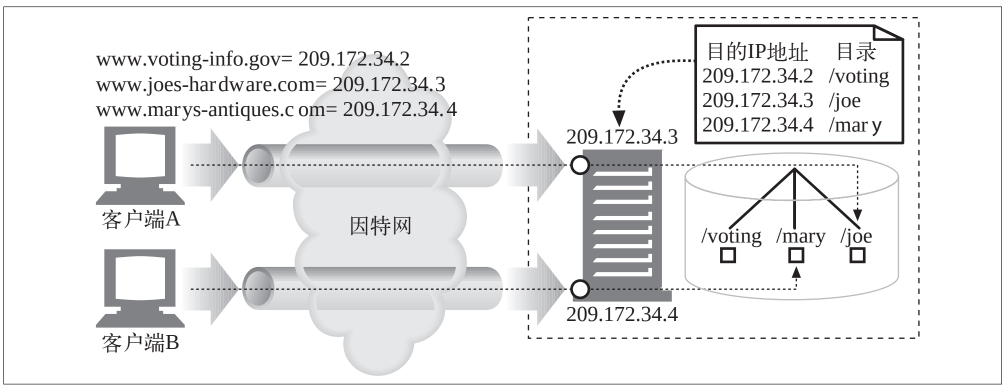


图 18-4　虚拟 IP 的主机托管

类似地，如果客户端 B 请求 http://www.marys-antiques.com/index.html。

- 客户端 B 查询 www.marys-antiques.com 的 IP 地址，得到 209.172.34.4。
- 客户端 B 打开到 Web 服务器的 TCP 连接，目的地址是 209.172.34.4。
- 客户端 B 发送请求，内容是 GET /index.html HTTP/1.0。
- Web 服务器判断出 209.172.34.4 是 Mary 的网站，根据 /mary 目录来完成请求，返回的是文件 /mary/index.html。 416

对大的托管者来说，虚拟 IP 的主机托管能够工作，但它会带来一些麻烦。

- 在计算机系统上能绑定的虚拟 IP 地址通常是有限制的。想在共享的服务器上托管成百上千的虚拟站点的服务商不一定能实现愿望。
- IP 地址是稀缺资源。有很多虚拟站点的托管者不一定能为被托管的网站获取足够多的 IP 地址。

- 托管者通过复制服务器来增加容量时，IP 地址短缺的问题就更严重了。随负载均衡体系的不同，可能会要求每个复制的服务器上有不同的虚拟 IP 地址，因此 IP 地址的需求量可能会随复制服务器的数量而倍增。

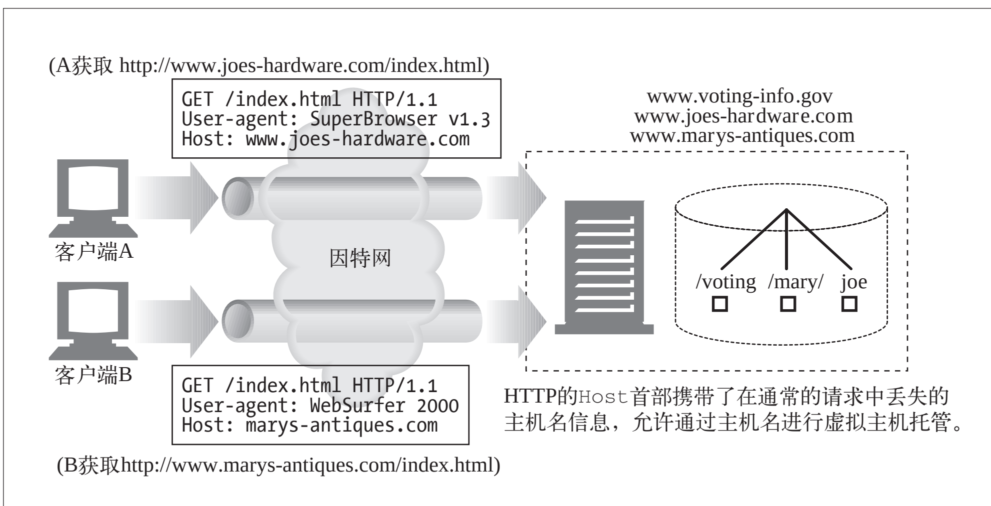
尽管虚拟 IP 的主机托管存在消耗地址的问题，但它仍然得到了广泛的运用。

4. 通过Host首部进行虚拟主机托管

为了避免过度的地址消耗和虚拟 IP 地址的限制，我们希望在虚拟站点间共享同一个 IP 地址，且仍能区分站点。但正如我们看到的那样，因为大多数浏览器只是把 URL 的路径发给服务器，关键的虚拟主机名信息被其丢掉了。

为了解决这个问题，浏览器和服务器的实现者扩展了 HTTP，把原始的主机名提供给服务器。不过，浏览器不能只发送完整的 URL，因为这会使许多只能接收路径的服务器无法工作。替代的方法是，把主机名（和端口号）放在所有请求的 `Host` 扩展首部中传送。

在图 18-5 中，客户端 A 和客户端 B 都发送了携带有要访问的原始主机名的 `Host` 首部。当服务器收到对 /index.html 的请求时，可以通过 `Host` 首部来判断要使用哪个资源。



417 图 18-5　用 `Host` 首部区分请求的虚拟主机

`Host` 首部最早是在 HTTP/1.0+ 中引入的，它是开发商实现的 HTTP/1.0 的扩展超集。遵循 HTTP/1.1 标准则必须支持 `Host` 首部。绝大多数现代浏览器和服务器都支持 `Host` 首部，但仍有一些客户端和服务器（以及网络机器人）不支持它。

18.2.3 HTTP/1.1的Host首部

Host 首部是 HTTP/1.1 的请求首部，定义在 RFC 2068 中。由于虚拟服务器的流行，绝大多数 HTTP 客户端（即使是不遵循 HTTP/1.1 的客户端），都实现了 Host 首部。

1. 语法与用法

Host 首部描述了所请求的资源所在的因特网主机和端口号，和原始的 URL 中得到的一样：

```
Host = "Host" ":" host [ ":" port ]
```

但要注意以下问题。

- 如果 Host 首部不包含端口，就使用地址方案中默认的端口。
- 如果 URL 中包含 IP 地址，Host 首部就应当包含同样的地址。
- 如果 URL 中包含主机名，Host 首部就必须包含同样的名字。
- 如果 URL 中包含主机名，Host 首部就不应当包含 URL 中这个主机名对应的 IP 地址，因为这样会扰乱虚拟主机托管服务器的工作，它在同一个 IP 地址上堆叠了很多虚拟站点。
- 如果 URL 中包含主机名，Host 首部就不应当包含这个主机名的其他别名，因为这样也会扰乱虚拟主机托管服务器的工作。
- 如果客户端显式地使用代理服务器，客户端就必须把*原始*服务器，而不是代理服务器的名字和端口放在 Host 首部中。以往，若干个 Web 客户端在启用客户端代理设置时，错误地把发出的 Host 首部设置成代理的主机名。这种错误行为会使代理和原始服务器都无法正常处理请求。
- Web 客户端必须在所有请求报文中包含 Host 首部。
- Web 代理必须在转发请求报文之前，添加 Host 首部。
- HTTP/1.1 的 Web 服务器必须用 400 状态码来响应所有缺少 Host 首部字段的 HTTP/1.1 请求报文。

下面是一段简单的 HTTP 请求报文，用于获取 www.joes-hardware.com 的主页，其中带有必需的 Host 首部字段： 418

```
GET http://www.joes-hardware.com/index.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/
png, */*
Accept-Encoding: gzip
Accept-Language: en
Host: www.joes-hardware.com
```

2. 缺失Host首部

有少量在用的老式浏览器不会发送 Host 首部。如果某个虚拟主机托管服务器使用 Host 首部来判断所服务的是哪个网站，而报文中没有出现 Host 首部的话，那它可能会把用户导向某个默认的 Web 页面（例如网络服务提供商的 Web 页面），也可能返回一个错误页面建议用户升级浏览器。

3. 解释Host首部

对于没有进行虚拟主机托管，而且不允许资源随请求主机的不同而变化的原始服务器来说，可以忽略 Host 首部字段的值。但资源会随主机名的不同而变化的原始服务器，都必须在一条 HTTP/1.1 请求判断其所请求的资源时使用下列规则。

（1）如果 HTTP 请求报文中的 URL 是绝对的（也就是说，包含方案和主机部分），就忽略 Host 首部的值。

（2）如果 HTTP 请求报文中的 URL 没有主机部分，而该请求带有 Host 首部，则主机 / 端口的值就从 Host 首部中取。

（3）如果通过第（1）步或第（2）步都无法获得有效的主机，就向客户端返回 400 Bad Request 响应。

4. Host首部与代理

某些版本的浏览器发送的 Host 首部不正确，尤其是配置使用代理的时候。例如，配置使用代理时，一些老版本的 Apple 和 PointCast 客户端会错误地把代理的名字，而不是原始服务器的名字放在 Host 首部里发送。

18.3 使网站更可靠

在下面列出的这些时间段内，网站通常是无法运作的。

- 服务器宕机的时候。
- 交通拥堵：突然间很多人都要看某个特别的新闻广播或涌向某个大甩卖网店。突然的拥堵可以使 Web 服务器过载，降低其响应速度，甚至使它彻底停机。
- 网络中断或掉线。

419 本节会展示一些预判和处理这些常见问题的方法。

18.3.1 镜像的服务器集群

服务器集群是一排配置相同的 Web 服务器，互相可以替换。每个服务器上的内容可以通过镜像复制，这样当某个服务器出问题的时候，其他的可以顶上。

镜像的服务器常常组成层次化的关系。某个服务器可能充当“内容权威”——它含有原始内容（可能就是内容作者上传的那个服务器）。这个服务器称为主原始服务器（master origin server）。从主原始服务器接收内容的镜像服务器称为复制原始服务器（replica origin server）。一种简单的部署服务器集群的方法是用网络交换机把请求分发给服务器。托管在服务器上的每个网站的 IP 地址就设置为交换机的 IP 地址。

在图 18-6 显示的镜像服务器集群中，主原始服务器负责把内容发送给复制原始服务器。对集群外部来说，内容所在的 IP 地址就是交换机的 IP 地址。交换机负责把请求发送到服务器上去。

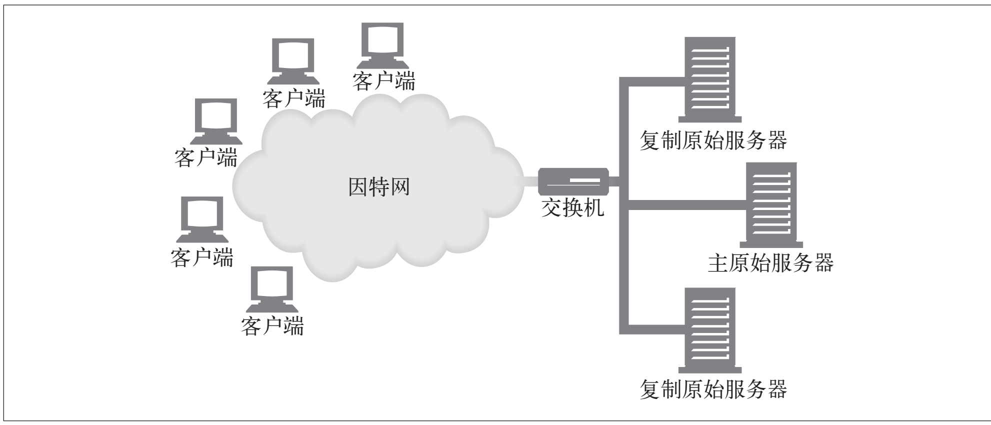


图 18-6　镜像的服务器集群

镜像 Web 服务器可以在不同的地点包含同样内容的副本。图 18-7 展示了 4 个镜像服务器，其中主服务器在芝加哥，复制服务器在纽约、迈阿密和小石城。主服务器为芝加哥地区的客户端服务，并肩负把内容传播给复制服务器的任务。

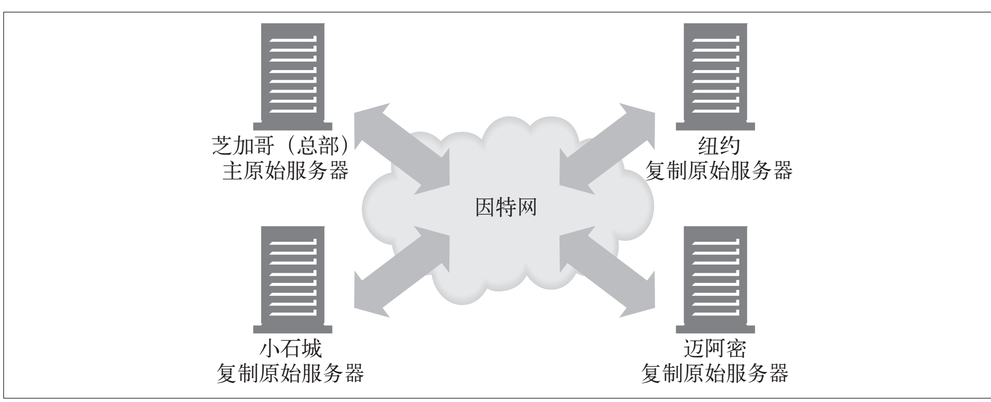


图 18-7　分散的镜像服务器

在图 18-7 的场景中，有以下两种方法把客户端的请求导向特定的服务器。

- HTTP 重定向
420 该内容的 URL 会解析到主服务器的 IP 地址，然后它会发送重定向到复制服务器。
- DNS 重定向
该内容的 URL 会解析到 4 个 IP 地址，DNS 服务器可以选择发送给客户端的 IP 地址。

请参见第 20 章，以获取详细信息。

18.3.2 内容分发网络

简单地说，内容分发网络（CDN）就是对特定内容进行分发的专门网络。这个网络中的节点可以是 Web 服务器、反向代理或缓存。

18.3.3 CDN中的反向代理缓存

在图 18-6 和图 18-7 中，复制原始服务器可以用反向代理（也称为替代物）缓存来代替。反向代理缓存可以像镜像服务器一样接受服务器请求。它们代表原始服务器中的一个特定集合来接收服务器请求。（根据内容所在的 IP 地址的广告方式，这是有可能的，原始服务器和反向代理缓存之间通常有协作关系，到特定的原始服务器的请求就由反向代理缓存来接收。）

反向代理和镜像服务器之间的区别在于反向代理通常是需求驱动的。它们不会保存原始服务器的全部内容副本，它们只保存客户端请求的那部分内容。内容在其高速缓存中的分布情况取决于它们收到的请求，原始服务器不负责更新它们的内容。为了更容易地访问“热点”内容（就是高请求率的内容），有些反向代理具有“预取”特性，可以在用户请求之前就从服务器上载入内容。

421 CDN 中带有反向代理时，可能会由于存在代理的层次关系而增加其复杂性。

18.3.4 CDN中的代理缓存

代理缓存也可以部署在类似图 18-6 和图 18-7 的环境中。与反向代理不同，传统的代理缓存能收到发往任何 Web 服务器的请求。（在代理缓存与原始服务器之间不需要有任何工作关系或 IP 地址约定。）但是与反向代理比起来，代理缓存的内容一般都是按需驱动的，不能指望它是对原始服务器内容的精确复制。某些代理缓存也可以预先载入热点内容。

按需驱动的代理缓存可以部署在其他环境中——尤其是拦截环境，在这种情况下，2 层或 3 层设备（交换机或路由器）会拦截 Web 流量并将其发送给代理缓存（参见图 18-8）。

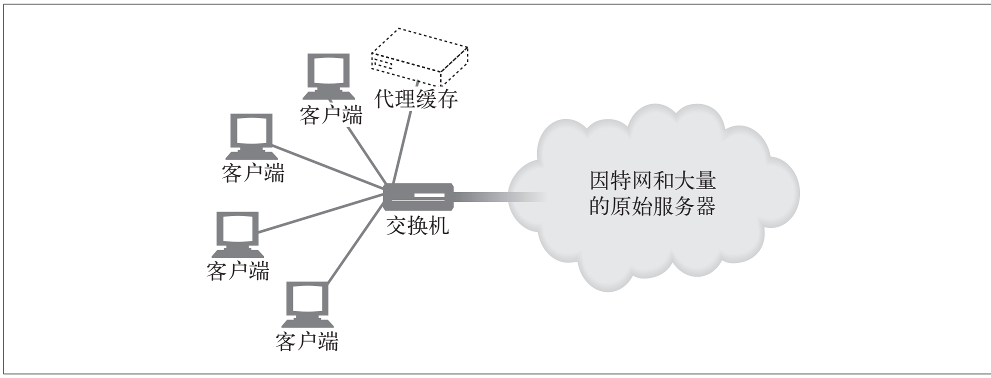


图 18-8　客户端的请求被交换机拦截并发给代理缓存

拦截环境依赖于在客户端和服务器之间设置网络的能力，这样，所有合适的 HTTP 请求才能真正发送到缓存中去。（参见第 20 章）。根据收到的请求，将内容分布在缓存中。

18.4　让网站更快

前面一节提到的很多技术也能帮助网站更快地加载。服务器集群和分布式代理缓存或反向代理服务器分散了网络流量，可以避免拥塞。分发内容使之更靠近终端用户，这样从服务器到客户端的传输时间就更短了。请求和响应穿过因特网，在客户端和服务器间传输的方式是影响资源访问速度最主要的因素。重定向方法的详细内容参见第 20 章。

加速网站访问的另一种方法是对内容进行编码以便更快地传输。比如，对内容进行压缩，但前提是接收的客户端能够把内容解压缩。请参见第 15 章了解更多细节。 422

18.5　更多信息

参阅第 3 部分以了解如何使 Web 站点安全。下面的因特网草案和文档提供了 Web 虚拟主机服务和内容分发的更多细节。

- http://www.ietf.org/rfc/rfc3040.txt
  RFC 3040，“Internet Web Replication and Caching Taxonomy”（“因特网 Web 复制和缓存分类法”），这份文档是关于 Web 复制与缓存应用术语的参考文献。

- http://search.ietf.org/internet-drafts/draft-ietf-cdi-request-routing-reqs-00.txt
“Request-Routing Requirements for Content Internetworking”（“内容网际互连的请求路由需求”）。

- *Apache: The Definitive Guide*[1]（《Apache 权威指南》）
Ben Laurie 和 Peter Laurie 著，O’Reilly & Associates 公司出版。这本书讲述如何
423 运行开源的 Apache Web 服务器。

注 1：本书影印版由人民邮电出版社出版。（编者注）

第19章

发布系统

怎样创建 Web 页面并放到 Web 服务器上去呢？在 Web 发展的"蛮荒"时代（比如 1995 年），可能要在文本编辑器中手工拼凑 HTML，用 FTP 手动把内容上传到 Web 服务器。这个过程很痛苦，很难与同事配合，也不是特别安全。

如今的发布工具使得创建、发布以及管理 Web 内容方便了许多。今天，用户可以交互式地编辑 Web 内容，在屏幕上看到它实际呈现的样子，轻轻点击一下就可以把内容发布到服务器，还能得到所有文件变化的通知。

许多支持远程发布内容的工具都使用了扩展的 HTTP 协议。本章将讲解以 HTTP 为基础的两种重要的 Web 内容发布技术：FrontPage 和 DAV。

19.1 FrontPage为支持发布而做的服务器扩展

FrontPage（FP）是微软公司提供的一种通用 Web 写作和发布工具包。FrontPage 的原始创意（FrontPage 1.0）是由维美尔（Vermeer）技术公司在 1994 年构思的，它是首个把网站管理和创建整合进一个统一工具的产品。微软公司 1996 年收购了维美尔公司，发行了 FrontPage 1.1。最新的版本——FrontPage 2002 版，是这条产品线上的第六代，是微软办公套件的核心组成部分。

19.1.1 FrontPage服务器扩展

作为"随处发布"战略的一部分，微软公司发布了一系列服务器端软件，称为"FrontPage 服务器扩展"，（FPSE）。这些服务器端组件和 Web 服务器集成在一起，
424 在网站和运行 FrontPage 的客户端（以及其他支持这些扩展的客户端）之间提供了必要的转接工作。

我们主要关注 FrontPage 客户端和 FPSE 之间的发布协议。该协议是对 HTTP 核心服务进行扩展而无需改变 HTTP 语义的一个设计范例。

FrontPage 的发布协议在 HTTP 的 POST 请求之上实现了一个 RPC（Remote Procedure Call，远程过程调用）层。它允许 FrontPage 客户端向服务器发送命令来更新网站上的文档、进行搜索以及在多个 Web 作者之间进行协作，等等。图 19-1 给出了这个通信过程的概貌。

Web 服务器看到以 FPSE（在非微软 IIS 服务器上就以一组 CGI 程序的方式实现）为接收地址的 POST 请求，就对其进行相应的引导。只要中间的防火墙和代理服务器都配置为允许使用 POST 方法，FrontPage 就能与服务器持续通信。

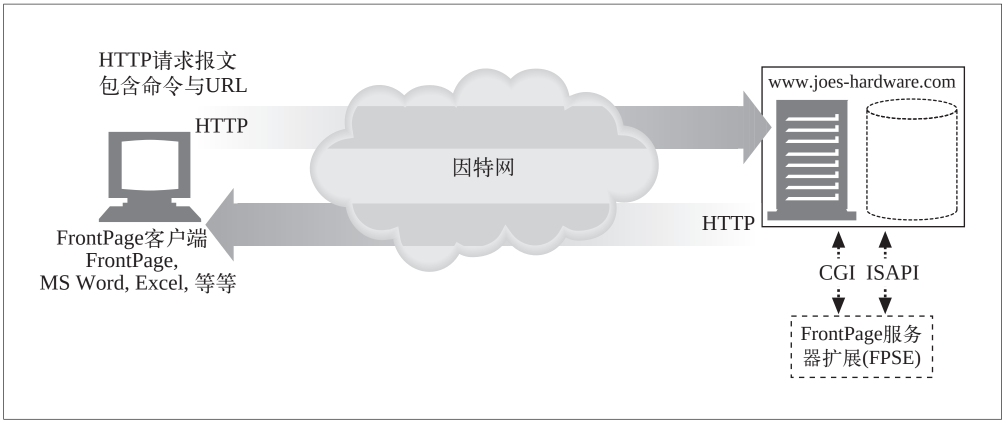


图 19-1　FrontPage 发布系统体系结构

19.1.2　FrontPage术语表

在我们深入研究 FPSE 定义的 RPC 层之前，先来了解一下其常用术语。

- 虚拟服务器

  在同一服务器上运行的多个网站之一，每个都有唯一的域名和 IP 地址。本质上说，虚拟服务器允许在单一的 Web 服务器上托管多个网站，在浏览器看来每个网站都像是由它自己专门的 Web 服务器托管的一样。支持虚拟服务器的 Web 服务器称为多路托管（multi-hosting）Web 服务器。配置有多个 IP 地址的机器称为多宿主（multi-homed）服务器（参阅 18.2 节以获取更多信息）。

- 根 Web

  Web 服务器默认的顶层内容目录，或者是在多路托管环境下，虚拟 Web 服务器的顶层内容目录。要访问根 Web，只需指定该服务器的 URL，而不需要指定页面名称。每个 Web 服务器只能有一个根 Web。 425

- 子 Web

  根 Web 的已命名子目录或另一个完全由 FPSE 扩展的子 Web。子 Web 可以是完全独立的实体，能够指定自己的管理和写作权限。此外，子 Web 还能提供方法（比如搜索）的作用范围。

19.1.3　FrontPage的RPC协议

FrontPage 客户端与 FPSE 使用专用的 RPC 协议来通信。该协议构建在 HTTP 的 POST 方法之上，它把 RPC 的方法及其相关的变量嵌入在 POST 请求的主体中。

在开始处理之前，客户端需要知道服务器上目标程序（FPSE 包中能够执行这些 POST 请求的相关部分）的位置和名称。接下来它会发送一个特殊的 GET 请求（参见图 19-2）。

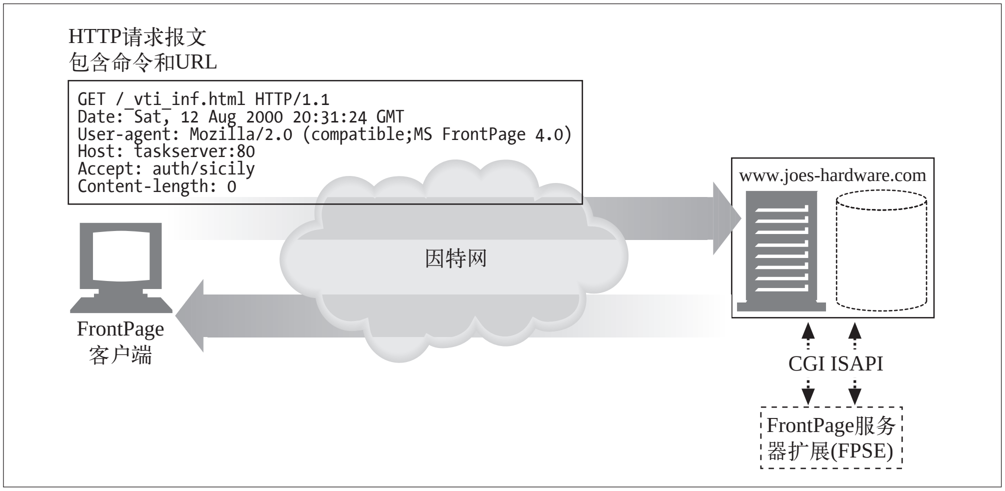


图 19-2　初始请求

得到返回的文件之后，FrontPage 客户端读取响应，寻找与 `FPShtmlScriptUrl`、`FPAuthorScriptUrl` 以及 `FPAdminScriptUrl` 相关的值。通常，这些值看起来是这样的：

```
FPShtmlScriptUrl="_vti_bin/_vti_rpc/shtml.dll"
FPAuthorScriptUrl="_vti_bin/_vti_aut/author.dll"
FPAdminScriptUrl="_vti_bin/_vti_adm/admin.dll"
```

`FPShtmlScriptUrl` 告诉客户端要执行“浏览时”命令（例如，获取 FPSE 的版本号）时应向哪里 POST 请求。

`FPAuthorScriptUrl` 告诉客户端要执行“写作时”命令时应向哪里 POST 请求。
426 类似地，`FPAdminScriptUrl` 告诉 FrontPage 向哪里发送管理操作的 POST 请求。

现在我们已知道这些程序所在的位置，可以发出请求了。

1. 请求

POST 请求的主体包含 RPC 命令，形式是 `method=<command>` 及任何需要的参数。例如，请求文档列表的 RPC 报文如下：

```
POST /_vti_bin/_vti_aut/author.dll HTTP/1.1
Date: Sat, 12 Aug 2000 20:32:54 GMT
```

```
User-Agent: MSFrontPage/4.0
..........................................

<BODY>
method=list+documents%3a4%2e0%2e2%2e3717&service%5fname=&listHiddenDoc
s=false&listExplorerDocs=false&listRecurse=false&listFiles=true&listFo
lders=true&listLinkInfo=true&listIncludeParent=true&listDerived=false
&listBorders=false&listChildWebs=true&initialUrl=&folderList=%5b%3bTW%
7c12+Aug+2000+20%3a33%3a04+%2d0000%5d
```

POST 方法的主体中含有发送给 FPSE 的 RPC 命令。与 CGI 程序一样，方法中的空格被编码为加号（+）字符。所有其他非字母数字的字符都被编码为 `%`*XX* 格式，*XX* 表示该字符的 ASCII 码。根据这种记号方式，更容易辨识的主体版本如下所示：

```
method=list+documents:4.0.1.3717
&service_name=
&listHiddenDocs=false
&listExplorerDocs=false
.....
```

其中某些元素的含义如下所述。

- `service_name`

  方法应该在该 URL 表示的网站上执行。必须是已有文件夹或者已有文件夹的下层文件夹。

- `listHiddenDocs`

  如果值为 true，就显示网站中隐藏的文档。所谓“隐藏”是指其 URL 的路径部分以“_”开头。

- `listExploreDocs`

  如果值为 true，就列出任务列表。

2. 响应

大多数 RPC 协议方法都有返回值。大多数常见的返回值都用来表示方法成功和各种错误。有些方法还有第三种类别的返回值，称为“采样返回码”。FrontPage 会对这些代码进行适当的解释，为用户提供准确的反馈。 427

继续讨论前面的例子，FPSE 处理这个 `listdocuments` 请求并返回必须的信息。示例响应如下：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 12 Aug 2000 22:49:50 GMT
Content-type: application/x-vermeer-rpc
```

```
X-FrontPage-User-Name: IUSER_MINSTAR

<html><head><title>RPC packet</title></head>
<body>
<p>method=list documents: 4.0.2.3717
<p>document_list=
<ul>
    <li>document_name=help.gif
<\ul>
```

可以从响应中看到，Web 服务器上可用文档的列表返回给了 FrontPage 客户端。在微软公司的网站上可以找到各种命令和响应的完整列表。

19.1.4 FrontPage的安全模型

任何直接访问 Web 服务器内容的发布系统都要非常注意其行为的潜在安全影响。FPSE 在极大程度上是依赖 Web 服务器来提供安全性的。

FPSE 安全模型定义了 3 种用户：管理员、作者以及浏览者，其中管理员拥有完全控制权。所有权限都是累积的，也就是说，所有的管理员都能编写和浏览 FrontPage 的网站。类似地，所有作者都有浏览权限。

对于给定的由 FPSE 扩展的网站，管理员、作者以及浏览者的列表都要定义好。所有的子 Web 可以从根 Web 继承权限，也可以自行定义。对于非 IIS 的 Web 服务器，所有的 FPSE 程序都要保存在标记为“可执行”的目录中（所有其他 CGI 程序也都有同样的限制）。Fpsrvadm，FrontPage 的服务器管理员实用工具，可以用来进行这种工作。在 IIS 服务器上，则可用 Windows 操作系统自身集成的安全模型。

在非 IIS 服务器上，Web 服务器的访问控制机制负责指定能够访问指定程序的用户。在 Apache 和 NCSA 的 Web 服务器上，访问控制文件名为 .htaccess；在 Netscape 服务器上，文件名是 .nsconfig。访问控制文件将用户、用户组以及 IP 地址与不同级别的权限关联起来：GET 是读权限，POST 是写权限，等等。例如，为了使用户在 Apache 的 Web 服务器上具有作者权限，.htaccess 文件应当允许该用户对 author.exe 进行 POST。这些访问规范文件常常是以目录为单位来定义的，这为权限定义提供
428 了极大的灵活性。

在 IIS 服务器上，权限是通过给定根 Web 目录或子 Web 的根目录上的 ACL（Access Control List，访问控制列表）来制定的。当 IIS 收到请求时，首先登录，并模拟用户，接着发送请求到上述三个 DLL（Dynamic Link Library，动态链接库）之一。收到请求的 DLL 根据目标文件夹上定义的 ACL 检查所扮演用户的证书。如果检查通过，请求的操作就由扩展 DLL 来执行。否则，就向客户端发回“permission denied”

（没有权限）报文。由于 IIS 和 Windows 操作系统的安全管理紧密集成在一起，所以可以使用用户管理器进行细粒度的控制。

尽管有着精密的安全模型，启用 FPSE 还是背负了带来显著安全风险的恶名。在多数情况下，这是由于网站管理员的粗心大意引起的。然而，早期版本的 FPSE 的确有严重的安全漏洞，从而引起了这种安全风险的担忧。完整地实现严密的安全模型困难重重，也加剧了这种担忧。

19.2 WebDAV与协作写作

WebDAV（Web Distributed Authoring and Versioning，分布式写作与版本管理）为 Web 发布增添了新的内容——协作。当前，最常见的协作实践（电子邮件或分布式文件共享）显然没有什么技术含量。这种实践很不方便还容易出错，而且几乎没有过程控制。请考虑运行一个为多国用户服务的汽车制造商的多语种网站。很容易看出，它非常需要安全可靠的带有发布原语和协作原语（比如锁定和版本管理等）的健壮系统。

WebDAV（作为 RFC 2518 发表）专注于对 HTTP 进行扩展，以提供协作写作的适宜平台。目前是在 IETF 的组织下，得到了许多厂商的支持，包括 Adobe、Apple、IBM、Microsoft、Netscape、Novell、Oracle 以及 Xerox。

19.2.1 WebDAV的方法

WebDAV 定义了一些新的 HTTP 方法并修改了其他一些 HTTP 方法的操作范围。WebDAV 新增的方法如下所述。

- PROPFIND
  获取资源的属性。
- PROPPATCH
  在一个或多个资源上设定一个或多个属性。
- MKCOL
  创建集合。
- COPY 429
  从指定的源端把资源或者资源集合复制到指定的目的地。目的地可以在另一台机器上。

- MOVE
  从指定的源端把资源或者资源集合移动到指定的目的地。目的地可以在另一台机器上。

- LOCK
  锁定一个或多个资源。

- UNLOCK
  把先前锁定的资源解锁。

  WebDAV 修改的 HTTP 方法有 DELETE、PUT 以及 OPTIONS。本章稍后将详细讨论新方法和修改后的方法。

19.2.2　WebDAV与XML

WebDAV 的方法通常都需要在请求和响应中关联大量的信息。HTTP 通常用报文首部来交换这类信息。然而，只在首部传输必要的信息已经暴露了一些局限性，包括难以有选择地对请求中的多个资源应用首部信息、不利于表示层次结构等。

WebDAV 借助了 XML 解决这个问题，它是一种元标记语言，提供了描述结构化数据的格式。XML 为 WebDAV 提供了以下解决方案。

- 对那些描述数据处理方式的指令进行格式化的方法。
- 在服务器上对复杂的响应进行格式化的方法。
- 交换与所处理的集合和资源有关的定制信息的方法。
- 承载数据自身的灵活工具。
- 对大多数国际化问题的健壮解决方案。

习惯上会把 XML 文档里引用的方案定义保存在一个 DTD（Document Type Definition，文档类型定义）文件中。因此，试图解释 XML 文档时，可以根据其中的 DOCTYPE 定义项得到和这份 XML 文档相关的 DTD 文件名。

WebDAV 定义了一个显式的 XML 名字空间——“DAV:”。简单地说，XML 的名字空间就是元素或属性的名字的集合。名字空间限定了嵌入的名字在域内必须是唯一的，这样就可以避免名字冲突。

WebDAV 规范（也就是 RFC 2518），定义了完整的 XML 方案。预定义的方案允许
430 解析软件不必读取 DTD 文件，而是根据预定义的 XML 方案来解释。

19.2.3　WebDAV首部集

WebDAV 的确引入了一些 HTTP 首部来增强新方法的功能。本节对其进行了简要介绍，请从 RFC 2518 中获取更多信息。新的首部如下所示。

- `DAV`

  用于了解服务器的 WebDAV 能力。WebDAV 支持的所有资源在响应 OPTIONS 请求时都要含有此首部。更多细节参见 19.2.14 节。

  ```
  DAV = "DAV" ":" "1" ["," "2"] ["," 1#extend]
  ```

- `Depth`

  这是一个关键元素，用于把 WebDAV 扩展到支持含有多级层次关系的资源组。参见 19.2.10 节以获取更多关于集合的详细解释。

  ```
  Depth = "Depth" ":" ("0" | "1" | "infinity")
  ```

  我们来看一个简单的例子。假设有个目录 DIR_A，其中有文件 file_1.html 和 file_2.html。如果某方法设置了 `Depth:0`，此方法就只作用到目录 DIR_A 自身；如果设置了 `Depth:1`，就作用到目录 DIR_A 及其包含的文件 file_1.html 和 file_2.html。

  `Depth` 首部对 WebDAV 定义的许多方法进行了修饰。用到 `Depth` 首部的方法有：LOCK、COPY 以及 MOVE。

- `Destination`

  定义这个首部是用来辅助 COPY 或 MOVE 方法标识目标 URI 的。

  ```
  Destination = "Destination" ":" absoluteURI
  ```

- `If`

  定义的唯一一个状态令牌是锁定令牌（参见 19.2.5 节）。`If` 首部定义了一组条件，如果这些条件都取值为非，请求就失败。类似 COPY 和 PUT 等方法可以在 `If` 首部中指定前置条件，使其有条件地适用。在实践中，最常见的需要满足的前置条件是先获得锁。

  ```
  If ="If"":" (1*No-tag-list | 1*Tagged-list)
  No-tag-list = List
  Tagged-list = Resource 1*List
  Resource = Coded-URL
  List = "(" 1*(["Not"](State-token | "[" entity-tag "]")) ")"
  State-token = Coded-URL
  Coded-URL = "<" absoluteURI ">"
  ```

- `Lock-Token`

  UNLOCK 方法需要用这个首部指定要删除的锁。LOCK 方法的响应中也有

Lock-Token 首部，载有关于锁定令牌的必须信息。

431
```
Lock-Token ="Lock-Token"":"Coded-URL
```

- Overwrite

  用于 COPY 和 MOVE 方法，指定是否要覆盖目标。参见本章后面关于 COPY 和 MOVE 方法的详细介绍。

```
Overwrite ="Overwrite"":" ("T" |"F")
```

- Timeout

  客户端用这个请求首部指定要求锁定的超时值。参见 19.2.5 节获取更多信息。

```
TimeOut = "Timeout" ":" 1#TimeType
TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Other)
DAVTimeOutVal = 1*digit
Other = "Extend" field-value
```

我们已经概述了 WebDAV 的意图及其实现，下面来仔细看看它提供的各种功能。

19.2.4 WebDAV的锁定与防止覆写

根据定义，协作要有不止一个人在给定的文档上工作。图 19-3 展示了和协作相关的固有问题。

在这个例子中，作者 A 和 B 联合编写一份规范。A 和 B 各自独立地对文档做了一些修改。A 把更新的文档上传到仓库。之后，B 也把自己的版本提交到仓库。不幸的是，由于 B 压根不知道 A 的修改，他没有把自己的版本与 A 的版本进行合并，从
432 而导致 A 的修改丢失。

为了改善这种问题，WebDAV 支持锁定的概念。但单靠锁定不能完全解决这个问题，还需要版本管理和消息传送才能提供完整的解决方案。

WebDAV 支持两种类型的锁：

- 对资源或集合的独占写锁；
- 对资源或集合的共享写锁。

独占写锁保证只有锁的拥有者有写权限。这种锁完全消除了潜在的冲突。共享写锁允许多个人在某个给定的文件上工作。这种锁定机制在多名作者对各自的活动都知晓的环境下可以很好地工作。WebDAV 通过 PROPFIND 方法提供了属性发现机制，可以判断对锁定的支持和所支持的锁定类型。

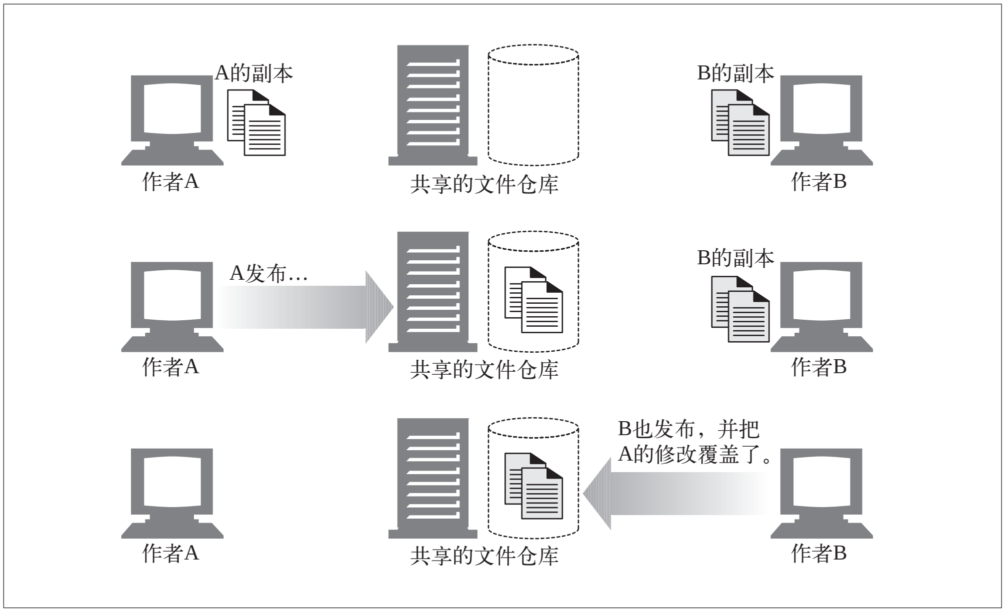


图 19-3　丢失更新问题

WebDAV 中有两个新方法支持锁定机制：LOCK 和 UNLOCK。

为了实现锁定，还需要有一种识别作者的机制。WebDAV 采用的是摘要认证（参见第 13 章）。

批准锁定时，服务器将域内唯一的令牌返回给客户端。与此相关的规范是 `opaquelocktoken` 锁定令牌 URI 方案。当客户端随后要执行写操作时，它连接到服务器并完成摘要认证步骤。一旦认证完成，WebDAV 客户端就发出带有锁定令牌的 PUT 请求。这样，只有正确的用户加上锁定令牌才可以完成写操作。

19.2.5　LOCK方法

WebDAV 中的一个强大特性是它能够允许单个 LOCK 请求锁定多个资源。WebDAV 的锁定不需要客户端保持与服务器的连接。

这是一个简单的 LOCK 请求示例： 433

```
LOCK /ch-publish.fm HTTP/1.1
Host: minstar
Content-Type: text/xml
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Content-Length: 201
```

```
<?xml version="1.0"?>
<a:lockinfo xmlns:a="DAV:">
     <a:lockscope><a:exclusive/></a:lockscope>
     <a:locktype><a:write/></a:locktype>
     <a:owner><a:href>AuthorA</a:href></a:owner>
</a:lockinfo>
```

提交的 XML 以 <lockinfo> 元素作为其基元素。在 <lockinfo> 结构中，有以下 3 种子元素。

- <locktype>

  指明锁定的类型。当前只有一种可选值，即 write。

- <lockscope>

  指明这是独占锁还是共享锁。

- <owner>

  这个字段设置为当前持有锁的人。

下面是这个 LOCK 请求的成功响应：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 20:56:18 GMT
Content-Type: text/xml
Content-Length: 419

<?xml version="1.0"?>
<a:prop xmlns:a="DAV:">
<a:lockdiscovery><a:activelock>
<a:locktype><a:write/></a:locktype>
<a:lockscope><a:exclusive/></a:lockscope>
<a:owner xmlns:a="DAV:"><a:href>AutherA</a:href></a:owner>
<a:locktoken><a:href>opaquelocktoken:*****</a:href></a:locktoken>
<a:depth>0</a:depth>
<a:timeout>Second-180</a:timeout>
</a:activelock></a:lockdiscovery>
</a:prop>
```

<lockdiscovery> 元素充当着存储锁信息的容器。嵌入在 <lockdiscovery> 元素中的子元素有 <activelock>，它持有请求发送来的信息（<locktype>、<lockscope> 以及 <owner>）。此外，<activelock> 中还含有以下子元素。

- <locktoken>

  用称为 opaquelocktoken 的 URI 方案唯一标识的锁。考虑到 HTTP 天生就是无状态的，该令牌用于在将来的请求中标识锁的所有权。

- <depth>

  它是 Depth 首部的值的副本。

- <timeout>

  指明锁的超时时间。在上面的响应中，超时值是 180 秒。

1. opaquelocktoken方案

opaquelocktoken 是设计用来在所有时间内对所有资源提供唯一令牌的方案。为了确保唯一性，WebDAV 规范规定采用 ISO-11578 中描述的 UUID 机制。 434

在实际实现的时候，有一定的回旋余地。服务器可以选择为每个 LOCK 请求生成一个 UUID，或者生成单个 UUID 并通过在结尾附加额外的字符来维护唯一性。从性能角度衡量，选择后面那种方法更好。不过，如果服务器选择实现后面那种方法，就必须保证附加的扩展部分永远不会重用。

2. XML元素<lockdiscovery>

XML 元素 <lockdiscovery> 提供了发现活跃的锁的机制。如果有人试图去给已经被锁定的文件上锁，他会收到指明当前拥有者的 XML 元素 <lockdiscovery>。<lockdiscovery> 元素列出了所有未解除的锁和相应的属性。

3. 锁的刷新和Timeout首部

为了刷新锁，客户端需要重新提交锁定请求，并把锁定令牌放在 If 首部中。返回的超时值可能和早先的超时值不同。

除了接受服务器给定的超时值，客户端也可以在 LOCK 请求中指明要求的超时值。这可以通过 Timeout 首部做到。Timeout 首部的语法允许客户端在逗号分隔的列表中描述一些选项。例如：

```
Timeout : Infinite, Second-86400
```

服务器没有义务必须满足这些选项。但是，客户端必须在 XML 元素 <timeout> 中提供锁定过期的时间。无论怎样，锁定超时只是一个指导值，服务器不一定受其约束。管理员可以手工重设，某些异常事件也可能导致服务器重设锁。客户端应当避免使锁定时间太长。

尽管有这些原语，图 19-3 中显示的“丢失更新问题”并没有得到完全解决。为了彻底解决这个问题，需要带有版本控制的协作事件系统。

19.2.6　UNLOCK方法

435 UNLOCK 方法用于解除资源上的锁。示例如下：

```
UNLOCK /ch-publish.fm HTTP/1.1
Host: minstar.inktomi.com
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Lock-Token:
opaquelocktoken:*********

HTTP/1.1 204 OK
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 20:56:18 GMT
```

与大多数资源管理请求一样，要使 UNLOCK 操作成功，WebDAV 要满足两个条件：第一，先前已经成功完成了摘要认证步骤；第二，要与在 `Lock-Token` 首部中发送的锁定令牌相匹配。

如果解锁成功，会向客户端发送 204 No Content 状态码。表 19-1 总结了 LOCK 和 UNLOCK 方法可能的状态码。

表19-1　LOCK和UNLOCK方法的状态码

状态码	定义者	方法	效果
200 OK	HTTP	LOCK	表明锁定成功
201 Created	HTTP	LOCK	表明通过创建该资源已成功锁定了不存在的资源
204 No Content	HTTP	UNLOCK	表明解锁成功
207 Multi-Status	WebDAV	LOCK	请求锁定多个资源。返回的资源状态码不完全一样，因此，这些状态码被封装在一个 207 响应中
403 Forbidden	HTTP	LOCK	表明客户端没有权限锁定资源
412 Precondition Failed	HTTP	LOCK	可能是随 LOCK 命令发送的 XML 中指明要满足某条件而服务器无法完成，也可能是无法强制执行锁定令牌
422 Unprocessable Property	WebDAV	LOCK	语义不适用——比如为不是集合的资源指定了非 0 的 `Depth`
423 Locked	WebDAV	LOCK	已处于锁定状态
424 Failed Dependency	WebDAV	UNLOCK	UNLOCK 指定了其他动作，并以它们的成功作为解锁的前提条件。如果无法成功完成这些有依赖关系的动作，就返回此错误码

19.2.7　属性和元数据

*属性*描述了资源的信息，包括作者的名字、修改日期、内容分级，等等。HTML 中的*元标记*的确提供了把这种信息嵌入在内容之中的机制，但很多种资源（比如所有二进制数据）都无法嵌入元数据。

像 WebDAV 这样的分布式协作系统对属性的需求就更复杂了。例如，考虑作者属性：当文档被编辑之后，应当更新这个属性以反映新的作者。WebDAV 专门把这种可动态修改的属性称为“活”属性。与之相对的是更长久的静态属性，比如 Content-Type，称为“死”属性。

为了支持查找和修改属性，WebDAV 扩展了 HTTP 以包括两个新方法：PROPFIND 和 PROPPATCH。后面几节给出了示例并讲解了相关的 XML 元素。 436

19.2.8 PROPFIND方法

PROPFIND 方法用于获取一个给定文件或一组文件（也称为“集合”）的属性。PROPFIND 支持 3 种类型的操作：

- 请求所有的属性及其值；
- 请求一组属性及其值；
- 请求所有属性的名称。

下面这个例子中，请求的是所有属性及其值：

```
PROPFIND /ch-publish.fm HTTP/1.1
Host: minstar.inktomi.com
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Depth: 0
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 0
```

<propfind> 请求元素指定了从 PROPFIND 方法返回的属性。下面的列表总结了用于 PROPFIND 请求的一些 XML 元素：

- <allprop>

  要求返回所有属性的名字和值。为了请求所有的属性及其值，WebDAV 客户端可以将 XML 子元素 <allprop> 作为 <propfind> 元素的一部分发送，或提交一个没有主体的请求。

- <propname>

  指定要返回属性名字的集合。

- <prop>

  <propfind> 元素的子元素。指定需要返回值的属性。例如：<a:prop> <a:owner />...... </a:prop>。

下面是对一个 PROPFIND 请求示例的响应： 437

```
HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0
...........

<?xml version="1.0"?>
<a:multistatusxmlns:b="urn:uuid:********/" xmlns:c="xml:"
xmlns:a="DAV:">
<a:response>
<a:href>http://minstar/ch-publish.fm </a:href>
<a:propstat>
<a:status>HTTP/1.1 200OK</a:status>
<a:prop>
<a:getcontentlength b:dt="int">1155</a:getcontentlength>
......................
......................
<a:ishidden b:dt="boolean">0</a:ishidden>
<a:iscollection b:dt="boolean">0</a:iscollection>
</a:prop>
</a:propstat>
</a:response></a:multistatus>
```

在这个例子中，服务器以 207 Multi-Status 状态码进行响应。WebDAV 将 207 用在 PROPFIND 和其他几个 WebDAV 方法上，它们同时作用在多个资源上并且每个资源可能有不同的响应。

响应中的几个 XML 元素的定义如下所示。

- `<multistatus>`
  多重响应的容器。

- `<href>`
  标识资源的 URI。

- `<status>`
  包含特定请求的 HTTP 状态码。

- `<propstat>`
  将一个 <status> 元素和一个 <prop> 元素组织在一起。<prop> 元素可以包含给定资源的一个或多个属性名 / 值对。

在上面列出的示例响应中，响应是针对 URI：http://minstar/ch-publish.fm 的。<propstat> 元素内嵌了一个 <status> 元素和一个 <prop> 元素。服务器为这个 URI 回复了一个 200 OK 响应，它定义在 <status> 元素中。<prop> 元素中有若干子元素，例子中只列出了一部分。

PROPFIND 的实例应用是对目录列表的支持。考虑到 PROPFIND 请求的表达能力，

单次调用就能获取集合的整个层次结构和其中各个独立实体的所有属性。

19.2.9 PROPPATCH方法

PROPPATCH 方法为对指定资源设置或删除多个属性提供了原子化机制。原子化可以保证要么所有请求都成功，要么跟所有请求都没发出一样。

PROPPATCH 方法的 XML 基元素是 <propertyupdate>。它作为一个容器使用，容纳了需要修改的属性。XML 的 <set> 和 <remove> 元素用于描述操作。

- <set>

  指定要设置的属性值。<set> 含有一个或多个 <prop> 子元素，它们依次包含了该资源上要设置的属性的名 / 值对。如果属性已存在，其值就被覆盖。 438

- <remove>

  指定要删除的属性。与 <set> 不同的是，在 <prop> 容器中只列出了属性的名称。

下面这个小例子设置并删除了 owner 属性：

```
<d:propertyupdate xmlns:d="DAV:" xmlns:o="http://name-space/scheme/">
  <d:set>
   <d:prop>
     <o:owner>Author A</o:owner>
   </d:prop>
  </d:set>

  <d:remove>
   <d:prop>
     <o:owner/>
   </d:prop>
  </d:remove>
</d:propertyupdate>
```

对 PROPPATCH 请求的响应和对 PROPFIND 请求的响应非常像。参见 RFC 2518 以获取更多信息。

表 19-2 总结了 PROPFIND 和 PROPPATCH 方法的状态码。

表19-2　PROPFIND与PROPPATCH方法的状态码

状　态　码	定　义　者	方　　法	效　　果
200 OK	HTTP	PROPFIND, PROPPATCH	命令成功
207 Multi-Status	WebDAV	PROPFIND, PROPPATCH	作用于一个或多个资源（或者集合）时，每个对象的状态都被封装到一个 207 响应中。这是一种常见的成功响应
401 Unauthorized	HTTP	PROPATCH	需要授权才能完成对属性的修改操作

（续）

状 态 码	定 义 者	方 法	效 果
403 Forbidden	HTTP	PROPFIND, PROPPATCH	对于 PROPFIND 来说，客户端不允许访问该属性。对于 PROPPATCH 来说，客户端不允许修改该属性
404 Not Found	HTTP	PROPFIND	属性不存在
409 Conflict	HTTP	PROPPATCH	与修改语义冲突——例如，试图修改只读的属性
423 Locked	WebDAV	PROPPATCH	目标资源被锁定，并且没有提供锁定令牌，或者锁定令牌不匹配
507 Insufficient Storage	WebDAV	PROPPATCH	没有足够的空间登记修改的属性

19.2.10　集合与名字空间管理

集合是指对预定义的层次结构中的资源进行的逻辑或物理上的分组。集合的一个典
439 型的例子就是目录。就像文件系统中的目录一样，集合作为其他资源（也包括其他集合，和文件系统中的目录一样）的容器使用。

WebDAV 使用了 XML 的名字空间机制。与传统的名字空间不同，XML 名字空间的分区在阻止所有名字空间冲突的同时，还允许进行精确的结构控制。

WebDAV 提供了 5 种方法对名字空间进行操作：DELETE、MKCOL、COPY、MOVE 以及 PROPFIND。本章前面已经讨论过 PROPFIND 了，下面来讨论其他方法。

19.2.11　MKCOL方法

MKCOL 方法允许客户端在服务器上指定的 URL 处创建集合。乍一看，仅仅为了创建集合而定义一个新方法好像有点儿多余。在 PUT 或 POST 方法之上加以修饰看起来就是个完美的替代方案。WebDAV 协议的设计者确实考虑过这些替代方案，但最终还是选择定义一个新方法。决策背后的一些理由如下所述。

- 为了使用 PUT 或 POST 来创建集合，客户端要随请求发送一些额外的“语义黏胶”。这当然是可以做到的，但定义这种特别的东西总是乏味且易错的。
- 大多数访问控制机制都是建立在方法类型之上的——只有少数能在库中创建和删除资源。如果给其他方法过多的功能，这些访问控制机制就无法运作了。

下面是请求的例子：

```
MKCOL /publishing HTTP/1.1
```

```
Host: minstar
Content-Length: 0
Connection: Keep-Alive
```

其响应可能是：

```
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 23:20:36 GMT
Location: http://minstar/publishing/
Content-Length: 0
```

我们再考察下面几种异常情况。

- 假设集合已经存在。如果发出 MKCOL /colA 请求而 colA 已存在（也就是说有名字空间冲突），请求会失败，状态码是 405 Method Not Allowed。
- 如果没有写权限，MKCOL 请求会得到 403 Forbidden 失败状态码。 440
- 如果发出 MKCOL /colA/colB 这样的请求而 colA 不存在，请求会失败，状态码是 409 Conflict。

创建了文件或集合之后，可以用 DELETE 方法来删除。

19.2.12　DELETE方法

我们已经在第 3 章探讨过 DELETE 方法了。WebDAV 扩展了它的语义以覆盖集合。

如果需要删除一个目录，就要提供 `Depth` 首部。如果没有指定 `Depth` 首部，DELETE 方法就假定 `Depth` 首部设定为无穷大——也就是说，该目录中的所有文件和子目录都会被删除。响应中也有 `Content-Location` 首部，其值就是刚被删除的集合。下面是一个请求的示例：

```
DELETE /publishing HTTP/1.0
Host: minstar
```

其响应的示例如下：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 14 May 2002 16:41:44 GMT
Content-Location: http://minstar/publishing/
Content-Type: text/xml
Content-Length: 0
```

删除集合时，总是有可能发生其中某个文件被其他人锁定而无法删除的情况。在这种情况下，集合自身也无法删除，服务器会以 207 Multi-Status 状态码响应。请求示例如下：

```
DELETE /publishing HTTP/1.0
Host: minstar
```

其响应的示例如下：

```
HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0
Content-Location: http://minstar/publishing/
..............
<?xml version="1.0"?>
<a:multistatus xmlns:a="DAV:">
<a:response>
<a:href>http://minstar/index3/ch-publish.fm</a:href>
<a:status> HTTP/1.1 423 Locked </a:status>
</a:response>
</a:multistatus>
```

在这次事务中，XML 元素 <status> 中含有状态码 423 Locked，表明资源 ch-
441 publish.fm 被别的用户锁定了。

19.2.13 COPY与MOVE方法

和 MKCOL 一样，有若干种方法可以定义新的 COPY 和 MOVE 操作方法。其中一种方式规定 COPY 方法先对源进行 GET 请求，下载资源，然后用 PUT 请求上传回服务器。可以设想，MOVE 方法也有类似的操作情况（有个额外的 DELETE 操作）。然而，这种处理过程无法很好地适应规模扩展——考虑一下在多级的集合上进行 COPY 或 MOVE 操作管理所涉及的问题吧。

COPY 和 MOVE 方法都将请求 URL 作为源，HTTP 的 Destination 首部的内容作为目标。MOVE 方法在 COPY 方法之外还要做一些工作：它把源 URL 复制到目的地，检查新创建的 URI 的完整性，再把源删除。请求示例如下：

```
{COPY,MOVE} /publishing HTTP/1.1
Destination: http://minstar/pub-new
Depth: infinity
Overwrite: T
Host: minstar
```

其响应示例如下：

```
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Wed, 15 May 2002 18:29:53 GMT
Location: http://minstar.inktomi.com/pub-new/
Content-Type: text/xml
Content-Length: 0
```

在对集合操作时，COPY 或 MOVE 的行为受到 Depth 首部的影响。如果没有

Depth 首部，就默认其值是无穷大（就是说，默认会把源目录的整个结构进行复制或移动）。如果 Depth 设置为 0，方法就只作用于资源本身。如果我们是对集合进行复制或移动的话，在目的地就只创建和源具有相同属性的集合——集合内部的成员就不再复制或移动了。

对于 MOVE 方法，Depth 首部的值只允许为无穷大，原因显而易见。

1. Overwrite首部的效果

COPY 和 MOVE 方法也可能使用 Overwrite 首部。Overwrite 首部的值可以是 T 或 F。如果设置为 T 而且目标已存在，就在 COPY 或 MOVE 之前，对目标资源执行 Depth 值为无穷大的 DELETE 操作。如果 Overwrite 标志设置为 F 而目标资源存在，则操作会失败。 442

2. 对属性的COPY/MOVE

当复制集合或元素时，默认会复制其所有属性。不过，请求可以带有可选的 XML 主体来提供额外的操作信息。可以指定要使操作成功，必须成功复制所有属性；或者定义要使操作成功，必须复制哪些属性。

下面有两个特殊状况下的例子。

- 假设把 COPY 或 MOVE 作用到 CGI 程序或者其他产生内容的脚本程序的输出上。为了保持语义，如果由 CGI 脚本产生的文件被复制或移动了，WebDAV 要提供 src 和 link 这两个 XML 元素，指向产生此页面的程序的位置。
- COPY 和 MOVE 方法不一定能够复制所有的活属性。例如，我们来看一个 CGI 程序。如果从 cgi-bin 目录中把它拷贝走，可能就不会再去执行它了。WebDAV 的现有规范让 COPY 和 MOVE 实现的是“尽力而为”解决方案，复制所有的静态属性和合适的活属性。

3. 被锁定的资源与COPY/MOVE

如果资源目前正被锁定，COPY 和 MOVE 都禁止把锁移动或复制到目标上。在这两种情况下，如果要在一个自己有锁的现存集合中创建目标，所复制或移动的资源就会被加到那个锁中。请看下面的例子：

```
COPY /publishing HTTP/1.1
Destination: http://minstar/archived/publishing-old
```

假设 /publishing 和 /archived 分别处于两个不同的锁之下：lock1 和 lock2。当 COPY

操作结束时，/publishing 仍旧处于 lock1 的范围内，而由于移动到了已被 lock2 锁定的集合中，publishing-old 被加入到了 lock2 中。如果是 MOVE 操作，就只有 publishing-old 被加入 lock2。

表 19-3 列出了 MKCOL、DELETE、COPY 以及 MOVE 方法最有可能碰到的状态码。

表19-3　MKCOL、DELETE、COPY和MOVE方法的状态码

状　态　码	定　义　者	方　　法	效　　果
102 Processing	WebDAV	MOVE、COPY	如果请求花费的时间超过 20 秒，服务器就发送这个状态码防止客户端超时。通常在 COPY 或 MOVE 大的集合时可以见到
201 Created	HTTP	MKCOL、COPY、MOVE	对于 MKCOL，表示集合创建成功。对于 COPY 和 MOVE，表示资源 / 集合已经复制或移动成功
204 No Content	HTTP	DELETE、COPY、MOVE	对于 DELETE，表示标准的成功响应。对于 COPY 和 MOVE，表示资源被成功地复制或移动而覆盖了已有的实体
207 Multi-Status	WebDAV	MKCOL、COPY、MOVE	对于 MKCOL，表示常见的成功响应。对于 COPY 和 MOVE 来说，如果有与资源相关（除请求 URI 之外的）的错误，服务器就回复 207 响应，其中带有详述错误的 XML 主体
403 Forbidden	HTTP	MKCOL、COPY、MOVE	对于 MKCOL，表明服务器不允许在指定的位置创建集合。对于 COPY 和 MOVE，表明源和目的是相同的
409 Conflict	HTTP	MKCOL、COPY、MOVE	三种情况类似，都是方法试图创建集合或资源，而中间集合不存在——例如，试图创建 colA/colB，而 colA 不存在
412 Precondition Failed	HTTP	COPY、MOVE	或者是 `Overwrite` 首部设置为 `F` 而目标存在，或者是 XML 主体描述了一个特定需求（比如保持 `liveness` 属性），而 COPY 或 MOVE 方法无法保持该属性
415 Unsupported Media Type	HTTP	MKCOL	服务器不支持或不理解如何创建请求的实体类型
422 Unprocessable Entity	WebDAV	MKCOL	服务器不理解请求中发送的 XML 主体
423 Locked	WebDAV	DELETE、COPY、MOVE	源或目标资源被锁定，或者方法提供的锁定令牌不匹配
502 Bad Gateway	HTTP	COPY、MOVE	目标在不同的服务器上并且缺少权限
507 Insufficient Storage	WebDAV	MKCOL、COPY	没有足够的空闲空间创建资源

443

19.2.14　增强的HTTP/1.1方法

WebDAV 修改了 HTTP 中 DELETE、PUT 以及 OPTIONS 方法的语义。GET 和 HEAD 方法的语义保持不变。POST 执行的操作总是由特定的服务器实现来定义的，而 WebDAV 没有对 POST 的语义进行任何修改。我们已经在 19.2.10 节讨论过 DELETE 方法了。这里将讨论 PUT 和 OPTIONS 方法。

1. PUT方法

尽管 PUT 不是由 WebDAV 定义的，但这是作者把内容传送到共享站点上的唯一方法。我们在第 3 章中讨论过 PUT 的一般功能。WebDAV 修改了该方法以支持锁定。 444

请看下面的例子：

```
PUT /ch-publish.fm HTTP/1.1
Accept: */*
If:<http://minstar/index.htm>(<opaquelocktoken:********>)
User-Agent: DAV Client (C)
Host: minstar.inktomi.com
Connection: Keep-Alive
Cache-Control: no-cache
Content-Length: 1155
```

为了支持锁定，WebDAV 在 PUT 请求中增加了 `If` 首部。在上面的事务中，`If` 首部的语义规定，如果 `If` 首部中说明的锁定令牌与资源（在这个例子中，是 `ch-publish.fm`）上的锁相匹配，就应当执行 PUT 操作。`If` 首部还用在其他一些方法中，比如 PROPPATCH、DELETE、MOVE、LOCK 以及 UNLOCK 等。

2. OPTIONS方法

我们在第 3 章中讨论过 OPTIONS。这通常是启用了 WebDAV 的客户端发出的第一个请求。客户端可以用 OPTIONS 方法验证 WebDAV 的能力。请看一个事务，其请求如下：

```
OPTIONS /ch-publish.fm HTTP/1.1
Accept: */*
Host: minstar.inktomi.com
```

其响应如下：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
MS-Author-Via: DAV
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE,
```

```
MKCOL,PROPFIND,PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, COPY, MOVE,
PROPFIND,PROPPATCH, SEARCH, LOCK, UNLOCK
```

在对 OPTIONS 方法的响应中有一些有趣的首部。下面的介绍略微打乱了一下顺序。

- `DAV` 首部携带了 DAV 遵从级别的信息。有下面两类遵从。

  第 1 类遵从

  要求服务器遵从 RFC2518 每节中的所有 MUST 需求。如果资源只能达到第 1 类遵从，就要在 `DAV` 首部中发送 1。

  第 2 类遵从

  满足所有第 1 类的需求，并增加对 LOCK 方法的支持。除了 LOCK 方法之外，第 2 类遵从还要求支持 `Timeout` 和 `Lock-Token` 首部以及 `<supportedlock>` 和 `<lockdiscovery>` 这两个 XML 元素。在 `DAV` 首部中的值 2 表明第 2 类遵从。

  在上面的例子中，`DAV` 首部表明这两类遵从都满足。

445

- `Public` 首部列出了这个特定的服务器支持的全部方法。
- `Allow` 首部通常包括 `Public` 首部所列方法的一个子集。它只列出了对这个特定资源（`ch-publish.fm`）有效的方法。
- `DASL` 首部说明了在 SEARCH 方法中使用的查询语法的类型。在这个例子中，就是 sql。更多关于 `DASL` 首部的细节参见 http://www.webdav.org。

19.2.15　WebDAV中的版本管理

这可能有点儿讽刺，尽管 DAV 这个名字中有 V，但版本管理这个特性却不是开始就有的。在多个作者的协作环境中，版本管理是至关重要的。实际上，为了完全解决更新丢失的问题（图 19-3 中有展示），锁定和版本化都是必不可少的。和版本管理相关的一些常见特征包括保存和访问以前的文档版本的能力、管理变更历史以及与变更相关的注解以详细说明变更过程的能力。

在 RFC 3253 中为 WebDAV 加入了版本管理功能。

19.2.16　WebDAV的未来发展

WebDAV 现在已经获得了良好的支持。可以工作的客户端实现包括 IE 5.x 及以上版本，Windows 系统的文件管理器以及微软的办公软件。在服务器端，可用的实现包括 IIS5.x 及以上版本，Apache 的 mod_dav 以及很多其他的系统。Windows XP 和 Mac OS 10.x 都提供了对 WebDAV 的原生支持。因此，为这些操作系统编写的任何

应用程序天生都具备使用 WebDAV 的能力。

19.3 更多信息

参考下列网址以获取更多信息。

- http://officeupdate.microsoft.com/frontpage/wpp/serk/
  微软 FrontPage 2000 服务器扩展资源包。

- http://www.ietf.org/rfc/rfc2518.txt?number=2518
  “HTTP Extensions for Distributed Authoring—WEBDAV”（“HTTP 的分布式写作扩展—WEBDAV”），作者是 Y. Goland、J. Whitehead、A. Faizi、S. Carter 以及 D. Jensen。 446

- http://www.ietf.org/rfc/rfc3253.txt?number=3253
  “Versioning Extensions to WebDAV”（“对 WebDAV 进行版本管理扩展”），作者是 G. Clemm、J. Amsden、T. Ellison、C. Kaler 和 J. Whitehead。

- http://www.ics.uci.edu/pub/ietf/Webdav/intro/Webdav_intro.pdf
  “WEBDAV: IETF Standard for Collaborative Authoring on the Web”（“WEBDAV：在 Web 上进行协同写作的 IETF 标准”），作者是 J. Whitehead 和 M. Wiggins。

- http://www.ics.uci.edu/~ejw/http-future/whitehead/http_pos_paper.html
  “Lessons from WebDAV for the Next Generation Web Infrastructure”（“在下一代 Web 的基础架构中借鉴 WebDAV”），作者是 J. Whitehead。

- http://www.microsoft.com/msj/0699/dav/davtop.htm
  “Distributed Authoring and Versioning Extensions for HTTP Enable Team Authoring”（“HTTP 中为支持团队写作而做的分布式写作和版本管理扩展”），作者是 L. Braginski 和 M.Powell。

- http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html
  “DAV Searching & Locating”（“DAV 中的搜索和定位”），作者是 S. Reddy、D. Lowry、S. Reddy、R. Henderson、J. Davis 以及 A. Babich。 447

第20章

重定向与负载均衡

HTTP 并不是独自运行在网上的。很多协议都会在 HTTP 报文的传输过程中对其数据进行管理。HTTP 只关心旅程的端点（发送者和接收者），但在包含有镜像服务器、Web 代理和缓存的网络世界中，HTTP 报文的目的地不一定是直接可达的。

本章介绍重定向技术，涉及网络工具、重定向技术细节以及判定 HTTP 报文最终目的地的协议。重定向技术通常可以用来确定报文是否终结于某个代理、缓存或服务器集群中某台特定的服务器。重定向技术可以将报文发送到客户端没有显式请求的地方去。

本章，我们会学习下列重定向技术，它们是如何工作的以及它们的负载均衡能力如何（如果有的话）：

- HTTP 重定向；
- DNS 重定向；
- 任播路由；
- 策略路由；
- IP MAC 转发；
- IP 地址转发；
- WCCP（Web 缓存协调协议）；
- ICP（缓存间通信协议）；
- HTCP（超文本缓存协议）；
- NECP（网元控制协议）；
- CARP（缓存阵列路由协议）；
- 448 WPAD（Web 代理自动发现协议）。

20.1 为什么要重定向

由于 HTTP 应用程序总是要做下列 3 件事情，所以在现代网络中重定向是普遍存在的：

- 可靠地执行 HTTP 事务；
- 最小化时延；
- 节约网络带宽。

出于这些原因，Web 内容通常分布在很多地方。这么做是出于可靠性的考虑。这样，如果一个位置出问题了，还有其他的可用；如果客户端能去访问较近的资源，就可以更快地收到所请求的内容，以降低响应时间；将目标服务器分散，还可以减少网络拥塞。可以将重定向当作一组有助于找到“最佳”分布式内容的技术。

由于重定向和负载均衡是共存的，所以本章也涵盖了负载均衡的话题。大多数重定向部署都包含了某些形式的负载均衡。也就是说，它们可以将输入报文的负载分摊到一组服务器中去。反之，因为输入报文一定会在分担负荷的服务器之间进行某种分布，所以任意形式的负载均衡中都包含了重定向。

20.2　重定向到何地

从客户端向目标发送 HTTP 请求，目标对其进行处理的角度来看，服务器、代理、缓存和网关对客户端来说都是服务器。很多重定向技术都可用于服务器、代理、缓存和网关，因为它们具有共同的，与服务器类似的特征。其他一些重定向技术是专门为特定类型的端点设计的，没有通用性。本章稍后的小节会介绍一些通用及专用的重定向技术。

Web 服务器会根据每个 IP 来处理请求。将请求分摊到复制的服务器中去，就意味着应该把对某特定 URL 的每条请求都发送到最佳的 Web 服务器上去（最靠近客户端的、或负载最轻的或采用其他优化策略选择的服务器）。重定向到某台服务器就像将所有需要给汽车加油的司机都送到最近的加油站去一样。

代理希望根据每个协议来处理请求。在理想情况下，某个代理附近的所有 HTTP 流量都应该通过这个代理传输。比如，如果某代理缓存靠近各种不同的客户端，那么理想情况下，所有请求都应流经这个代理缓存，因为代理缓存上会存储常用的文档，可以直接提供，从而避免通过更长、更昂贵的路径连接到原始服务器。重定向到代理就像从一条主要通路（无论它通往何处）上将流量分流到一条本地快捷路径上去 449
一样。

20.3　重定向协议概览

重定向的目标是尽快地将 HTTP 报文发送到可用的 Web 服务器上去。在穿过因特网的路径上，HTTP 报文传输的方向会受到 HTTP 应用程序和报文经由的路由设备的影响，参见以下示例。

- 配置创建客户端报文的浏览器应用程序，使其将报文发送给代理服务器。
- DNS 解析程序会选择用于报文寻址的 IP 地址。对不同物理地域的不同客户端来说，这个 IP 地址可能不同。
- 报文经过网络传输时，会被划分为一些带有地址的分组；交换机和路由器会检查分组中的 TCP/IP 地址，并据此来确定分组的发送路线。
- Web 服务器可以通过 HTTP 重定向将请求反弹给不同的 Web 服务器。

浏览器配置、DNS、TCP/IP 路由以及 HTTP 都提供了重定向报文的机制。注意，有些方法，比如浏览器配置，只有在将流量重定向到代理的时候才有意义，而其他一些方法（比如 DNS 重定向），则可用于将流量发送给任意服务器。[1]

表 20-1 总结了用来将报文重定向到服务器的重定向方法，本章稍后逐一讨论每种方法。

表20-1　通用的重定向方法

机　　制	工作方式	重新路由的基础	局　限　性
HTTP 重定向	最初，HTTP 请求先到第一台 Web 服务器，这台服务器会选择一台“最佳”的 Web 服务器为其提供内容。第一台 Web 服务器会向客户端发送一条到指定服务器的 HTTP 重定向。客户端会将请求重新发送到选中的服务器上	选择最短路径时可用的选项很多，包括轮转（round-robin）负载均衡和最小化时延等	可能会很慢——每个事务都包含了附加的重定向步骤。而且，第一台服务器一定要能够处理请求负载
DNS 重定向	DNS 服务器决定在 URL 的主机名中返回多个 IP 地址中的哪一个	选择最短路径时可用的选项很多，包括轮转（round-robin）负载均衡和最小化时延等	需要配置 DNS 服务器
任播寻址	几台服务器使用相同的 IP 地址。每台服务器都会伪装成一个骨干路由器。其他路由器会将共享 IP 地址分组发送给最近的服务器（认为它们将分组发送给最近的路由器）	路由器有内建的最短路径路由功能	需要拥有 / 配置路由器。有地址冲突的风险。如果路由变化了，与已建立的 TCP 连接相关的分组会被发送到其他的服务器，可能会使 TCP 连接断裂
IP MAC 转发	交换机或路由器这样的网元会读取分组的目的地址。如果应该将分组重定向，交换机会将服务器或代理的目标 MAC 地址赋予分组	节省带宽，提高 QOS。负载均衡	服务器或代理的跳距必须是 1
IP 地址转发	第四层交换机会评估分组的目的端口并将重定向分组的 IP 地址改成代理或镜像服务器的 IP 地址	节省带宽，提高 QOS。负载均衡	服务器或代理可能看不到真正的客户端 IP 地址

450

注 1：DNS 也不能随便重定向，目标 IP 地址的服务器上没有托管所需的虚拟主机的话，就和浏览器代理配错一样，毫无意义。（译者注）

表 20-2 总结了将报文重定向到代理服务器的重定向方法。

表20-2　代理与缓存重定向技术

机　　制	工作方式	重新路由的基础	局　限　性
显式浏览器配置	配置 Web 浏览器，使其将 HTTP 报文发送给附近的一个代理，通常是缓存。可以由终端用户或管理浏览器的服务进行配置	节省带宽，提高 QoS。负载均衡	取决于配置浏览器的能力
代理自动配置（PAC）	Web 浏览器从配置服务器中解析出 PAC 文件。这个 PAC 文件会告诉浏览器为每个 URL 使用什么代理	节省带宽，提高 QoS。负载均衡	必须配置浏览器，使其去查询配置服务器
Web Proxy 代理自动发现协议（WPAD）	Web 浏览器向配置服务器查询一个 PAC 文件的 URL。与单独使用 PAC 不同，不需要将浏览器配置为使用特定的配置服务器	配置服务器，将 URL 建立在客户端 HTTP 请求首部提供的信息之上。负载均衡	只有部分浏览器支持 WPAD
Web 缓存协调协议（WCCP）	路由器会评估一个分组的目的地址，并用代理或镜像服务器的 IP 地址将重定向分组封装起来。可以与很多现有路由器共同工作。可以将分组封装起来，这样客户端的 IP 地址就不会丢失了	节省带宽，提高 QoS。负载均衡	必须使用支持 WCCP 的路由器。有些拓扑结构方面的限制
因特网缓存协议（ICP）	代理缓存会在一组兄弟代理缓存中查询所请求的内容。还支持缓存的分层结构	从兄弟代理或父代理缓存中获取内容比从原始服务器中获取更快	请求内容时只使用了 URL，所以会降低缓存命中率
缓存分组路由协议（CARP）	一种代理缓存散列协议。允许缓存将请求转发给一个父缓存。与 ICP 不同的是，高速缓存上的内容是不相交的，这组缓存会像一个大型缓存那样工作	从附近的对等高速缓存中获取内容要比从原始服务器上获取快	CARP 无法支持兄弟关系。所有 CARP 客户端都必须在配置上达成一致；否则，不同的客户端就会向不同的父代理缓存发送相同的 URI，降低命中率 451
超文本缓存协议（HTCP）	参与的代理缓存可以向一组兄弟缓存查询所请求的内容。支持 HTTP 1.0 和 1.1 首部，以便精细地调整缓存查询	从兄弟代理或父代理缓存中获取内容比从原始服务器上获取快	

20.4 通用的重定向方法

本节我们会深入介绍服务器和代理常用的各种重定向方法。可以通过这些技术将流量重定向到不同的（可能更优的）服务器，或者通过代理来转发流量。具体来说，我们会介绍 HTTP 重定向、DNS 重定向、任播寻址、IP MAC 转发以及 IP 地址转发。

20.4.1 HTTP重定向

Web 服务器可以将短的重定向报文发回给客户端，告诉他们去其他地方试试。有些 Web 站点会将 HTTP 重定向作为一种简单的负载均衡形式来使用。处理重定向的服务器（重定向服务器）找到可用的负载最小的内容服务器，并将浏览器重定向到那台服务器上去。对广泛分布的 Web 站点来说，确定"最佳"的可用服务器会更复杂一些，不仅要考虑到服务器的负载，还要考虑到浏览器和服务器之间的因特网距离。与其他一些形式的重定向相比，HTTP 重定向的优点之一就是重定向服务器知道客户端的 IP 地址；理论上来讲，它可以做出更合理的选择。

下面是 HTTP 重定向的工作过程。在图 20-1a 中，Alice 向 www.joes-hardware.com 发送了一条请求：

```
GET /hammers.html HTTP/1.0
Host: www.joes-hardware.com
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
```

在图 20-1b 中，服务器没有回送带有 HTTP 状态码 200 的 Web 页面主体，而是回送了一个带有状态码 302 的重定向报文：

```
HTTP/1.0 302 Redirect
Server: Stronghold/2.4.2 Apache/1.3.6
Location: http://161.58.228.45/hammers.html
```

现在，在图 20-1c 中，浏览器会用重定向 URL 重新发送请求，这次会发送给主机 161.58.228.45：

```
GET /hammers.html HTTP/1.0
Host: 161.58.228.45
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
```

另一个客户端可能会被重定向到另一台服务器上去。在图 20-1d-f 中，Bob 的请求会
452 被重定向到 161.58.228.46。

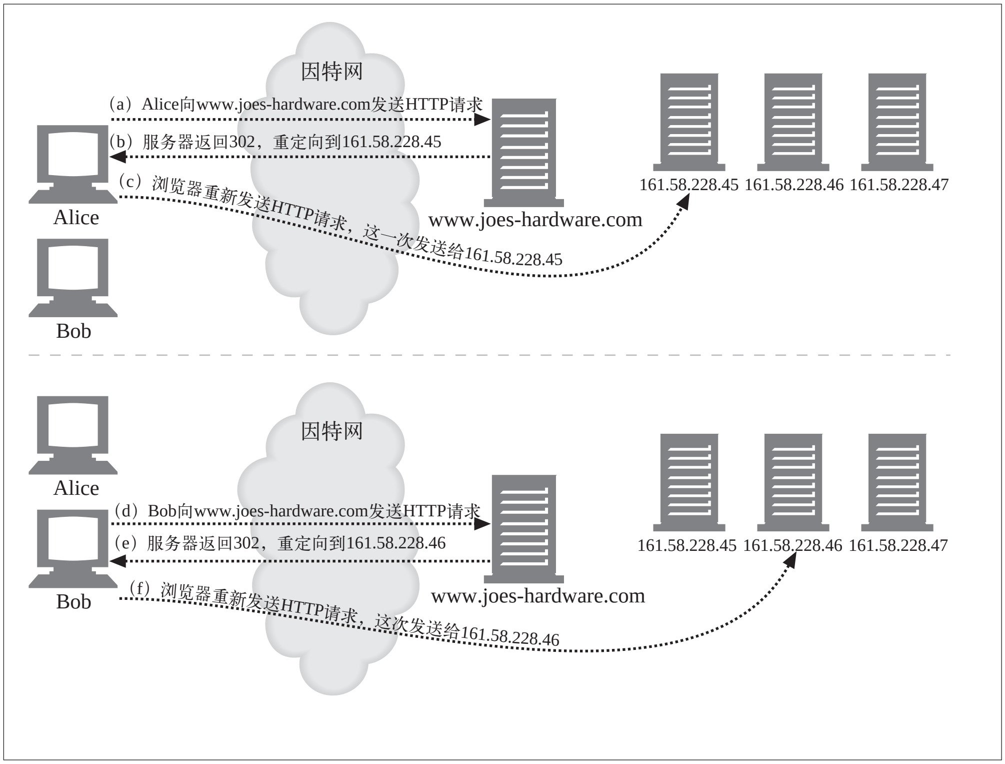


图 20-1　HTTP 重定向

HTTP 重定向可以在服务器间导引请求，但它有以下几个缺点。

- 需要原始服务器进行大量处理来判断要重定向到哪台服务器上去。有时，发布重定向所需的处理量几乎与提供页面本身所需的处理量一样。
- 增加了用户时延，因为访问页面时要进行两次往返。
- 如果重定向服务器出故障，站点就会瘫痪。

由于存在这些弱点，HTTP 重定向通常都会与其他一种或多种重定向技术结合使用。

20.4.2　DNS重定向

每次客户端试图访问 Joe 的五金商店的网站时，都必须将域名 www.joes-hardware.com 解析为 IP 地址。DNS 解析程序可能是客户端自己的操作系统，可能是客户端网络中的一台 DNS 服务器，或者是一台远距离的 DNS 服务器。DNS 允许将几个 IP 地址关联到一个域中，可以配置 DNS 解析程序，或对其进行编程，以返回可变的 IP 地址。解析程序返回 IP 地址时所基于的原则可以很简单（轮转），也可以很复杂（比如查看几台服务器上的负载，并返回负载最轻的服务器的 IP 地址）。 453

在图 20-2 中，Joe 为 www.joes-hardware.com 运行了 4 台服务器。DNS 服务器要决定为 www.joes-hardware.com 返回 4 个 IP 地址中的哪一个。最简单的 DNS 决策算法就是轮转。

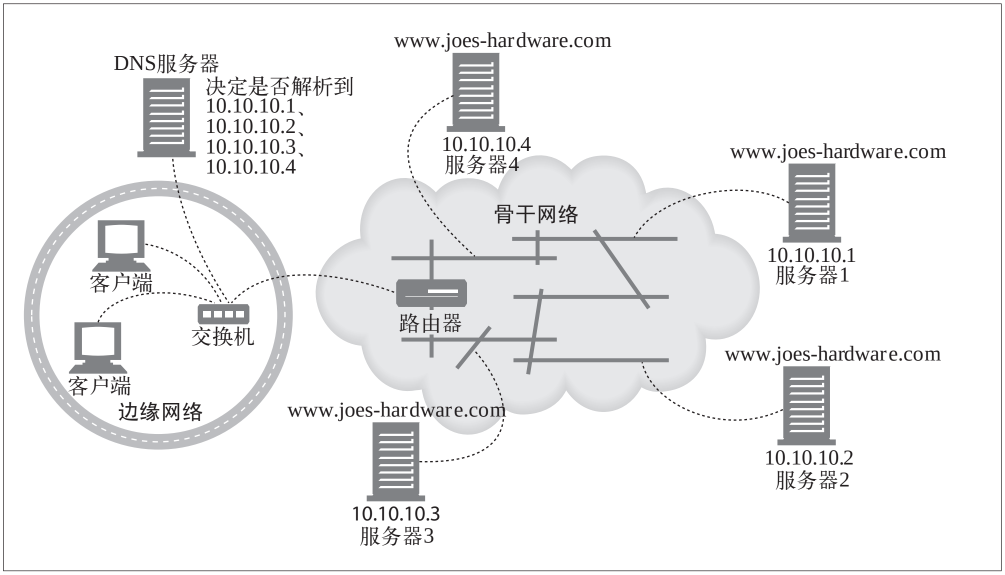


图 20-2　基于 DNS 的重定向

整个 DNS 解析过程的完整介绍，请参见本章末尾列出的 DNS 参考资料。

1. DNS轮转

最常见的重定向技术之一也是最简单的重定向技术之一。DNS 轮转使用了 DNS 主机名解析中的一项特性，在 Web 服务器集群中平衡负载。这是一种单纯的负载均衡策略，没有考虑任何与客户端和服务器的相对位置，或者服务器当前负载有关的因素。

我们来看看 CNN.com 实际上都做了些什么。我们在 2000 年 5 月初，用 Unix 中的工具 nslookup 来查找与 CNN.com 相关的 IP 地址。例 20-1 给出了结果。[2]

例 20-1　www.cnn.com 的 IP 地址

454

```
% nslookup www.cnn.com
Name: cnn.com
Addresses: 207.25.71.5, 207.25.71.6, 207.25.71.7, 207.25.71.8
207.25.71.9, 207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23
207.25.71.24, 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28
```

注 2：DNS 的结果是在 2000 年 5 月 7 日，从北加州解析出来的。有些特定值可能会随时间发生变化，有些 DNS 系统还会根据客户端的位置返回不同的值。

```
207.25.71.29, 207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245
207.25.71.246
Aliases: www.cnn.com
```

网站 www.cnn.com 实际上是 20 个不同的 IP 地址组成的集群。每个 IP 地址通常都意味着一台不同的物理服务器。

2. 多个地址及轮转地址的循环

大多数 DNS 客户端只会使用多地址集中的第一个地址。为了均衡负载，大多数 DNS 服务器都会在每次完成查询之后对地址进行轮转。这种地址轮转通常称作 DNS 轮转。

例如，对 www.cnn.com 进行三次连续的 DNS 查找可能会返回例 20-2 给出的那种 IP 地址轮转列表。

例 20-2　轮转 DNS 地址列表

```
% nslookup www.cnn.com
Name: cnn.com
Addresses: 207.25.71.5, 207.25.71.6, 207.25.71.7, 207.25.71.8
207.25.71.9, 207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23
207.25.71.24, 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28
207.25.71.29, 207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245
207.25.71.246

% nslookup www.cnn.com
Name: cnn.com
Addresses: 207.25.71.6, 207.25.71.7, 207.25.71.8, 207.25.71.9
207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23, 207.25.71.24
207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28, 207.25.71.29
207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245, 207.25.71.246
207.25.71.5

% nslookup www.cnn.com
Name: cnn.com
Addresses: 207.25.71.7, 207.25.71.8, 207.25.71.9, 207.25.71.12
207.25.71.20, 207.25.71.22, 207.25.71.23, 207.25.71.24, 207.25.71.25
207.25.71.26, 207.25.71.27, 207.25.71.28, 207.25.71.29, 207.25.71.30
207.25.71.82, 207.25.71.199, 207.25.71.245, 207.25.71.246, 207.25.71.5
207.25.71.6
```

在例 20-2 中：

- 第一次 DNS 查找时的第一个地址为 207.25.71.5；
- 第二次 DNS 查找时的第一个地址为 207.25.71.6；
- 第三次 DNS 查找时的第一个地址为 207.25.71.7。 455

3. 用来平衡负载的DNS轮转

由于大多数 DNS 客户端只使用第一个地址，所以 DNS 轮转可以在多台服务器间提供负载均衡。如果 DNS 没有对地址进行轮转，大部分客户端就总是会将负载发送给第一台服务器。

图 20-3 说明了 DNS 轮转循环是如何平衡负载的。

- Alice 试图连接 www.cnn.com 时，会用 DNS 查找 IP 地址，得到 207.25.71.5 作为第一个 IP 地址。在图 20-3c 中，Alice 连接到 Web 服务器 207.25.71.5。
- Bob 随后试图连接 www.cnn.com 时，也会用 DNS 查找 IP 地址，但由于地址列表在 Alice 上次请求的基础上轮转了一个位置，所以他会得到一个不同的结果。Bob 得到 207.25.71.6 作为第一个 IP 地址，在图 20-3f 中它连接到了这台服务器上。

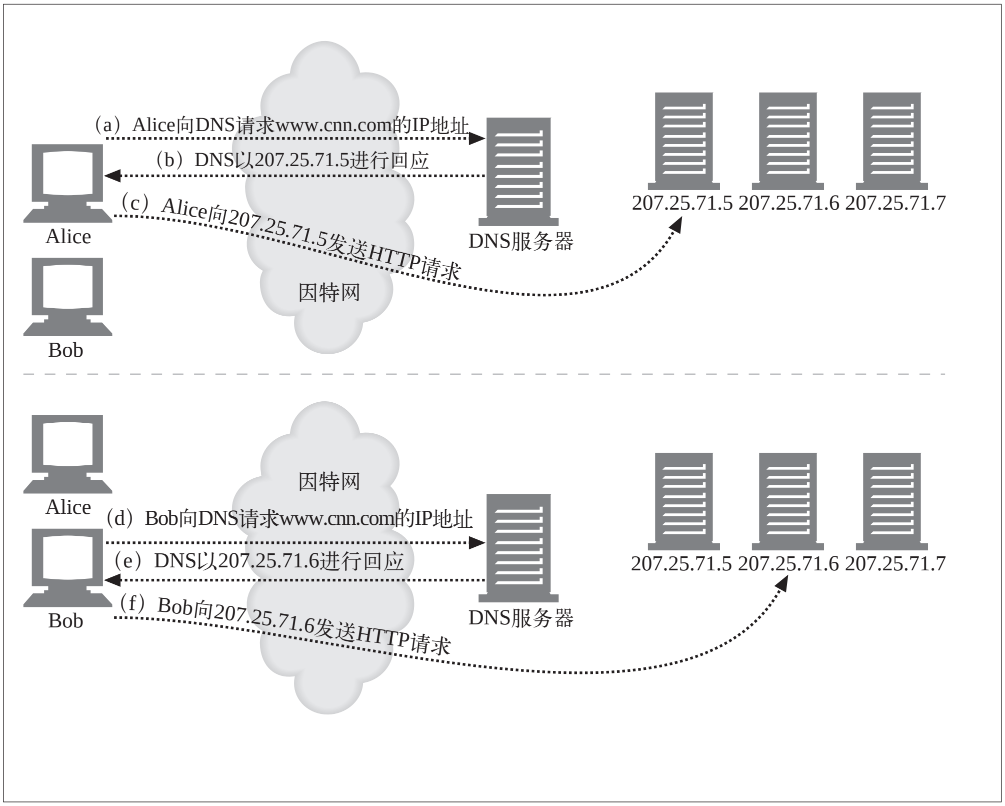


图 20-3　DNS 轮转在服务器集群的各台服务器之间进行负载均衡

4. DNS缓存带来的影响

DNS 对服务器的每次查询都会得到不同的服务器地址序列，所以 DNS 地址轮转会将负载分摊。但是这种负载均衡并不完美，因为 DNS 查找的结果可能会被记住，并被各种应用程序、操作系统和一些简易的子 DNS 服务器重用。很多 Web 浏览器都 456 会对主机进行 DNS 查找，然后一次次地使用相同的地址，以减少 DNS 查找的开销，而且有些服务器也更愿意保持与同一台客户端的联系。另外，很多操作系统都会自动进行 DNS 查找，并将结果缓存，但并不会对地址进行轮转。因此，DNS 轮转通常都不会平衡单个客户端的负载——一个客户端通常会在很长时间内连接到一台服务器上。

尽管 DNS 没有对单个客户端的事务进行跨服务器副本的处理，但在分散多个客户端的总负荷方面它做得相当好。只要有大量具有相同需求的客户端，就可以将负载合理地分散到各个服务器上去。

5. 其他基于DNS的重定向算法

我们已经讨论了 DNS 是如何对每条请求进行地址列表轮转的。但是，有些增强的 DNS 服务器会使用其他一些技术来选择地址的顺序。

- *负载均衡算法*

  有些 DNS 服务器会跟踪 Web 服务器上的负载，将负载最轻的 Web 服务器放在列表的最前面。

- *邻接路由算法*

  Web 服务器集群在地理上分散时，DNS 服务器会尝试着将用户导向最近的 Web 服务器。

- *故障屏蔽算法*

  DNS 服务器可以监视网络的状况，并将请求绕过出现服务中断或其他故障的地方。

通常，运行复杂服务器跟踪算法的 DNS 服务器就是在内容提供者控制之下的一个权威服务器（参见图 20-4）。

有一些分布式主机服务会使用这个 DNS 重定向模型。对于那些要查找附近服务器的服务来说，这个模型的一个缺点就是，权威 DNS 服务器只能用本地 DNS 服务器的 IP 地址，而不能用客户端的 IP 地址来做决定。

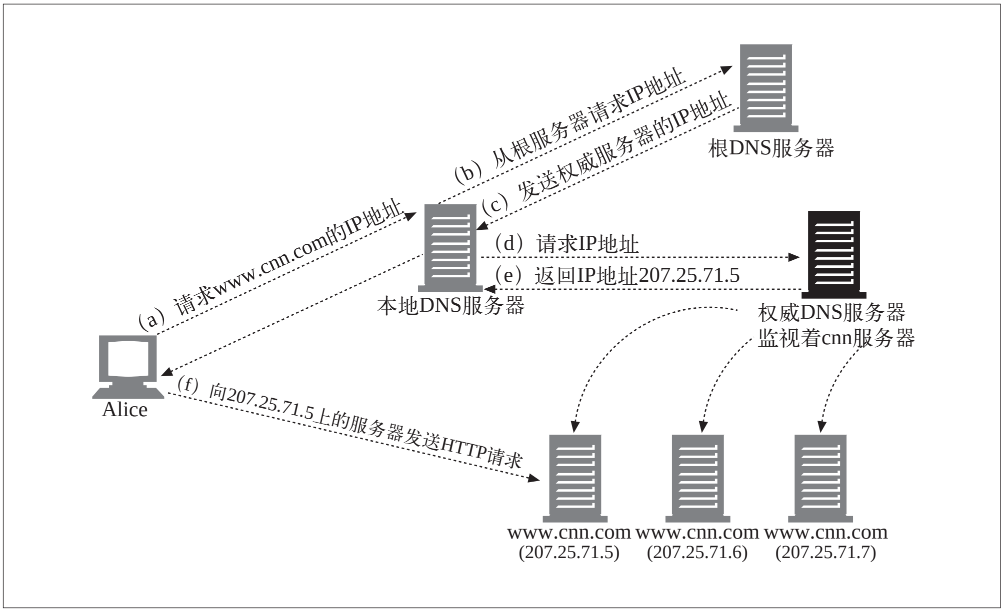


图 20-4　涉及权威服务器的 DNS 请求

20.4.3　任播寻址

在任播寻址中，几个地理上分散的 Web 服务器拥有完全相同的 IP 地址，而且会通过骨干路由器的“最短路径”路由功能将客户端的请求发送给离它最近的服务器。要使这种方法工作，每台服务器都要向邻近的骨干路由器广告，表明自己是一台路
457 由器。Web 服务器会通过路由器通信协议与其邻近的骨干路由器通信。骨干路由器收到发送给任播地址的分组时，会（像平常一样）寻找接受那个 IP 地址的最近的“路由器”。由于服务器是将自己作为那个地址的路由器广告出去的，所以骨干路由器会将分组发送给服务器。

在图 20-5 中，三台服务器为同一个 IP 地址 10.10.10.1 服务。洛杉矶（LA）服务器将此地址广告给 LA 路由器，纽约（NY）服务器同样将此地址广告给 NY 路由器，以此类推。服务器会通过路由器协议与路由器进行通信。路由器会将目标为 10.10.10.1 的客户端请求自动地转发到广告这个地址的最近的服务器上去。在图 20-5 中，对 IP 地址 10.10.10.1 的请求会被转发给服务器 3。

任播寻址仍然是项实验性技术。要使用分布式任播技术，服务器就必须“使用路由器语言”，而且路由器必须能够处理可能出现的地址冲突，因为因特网地址基本上都是假定一台服务器只有一个地址的。（如果没有正确地实现，可能会造成很严重的“路由泄露”问题。）分布式任播是一种新兴技术，可以为那些自己控制骨干网络的

内容提供商提供一种解决方案。

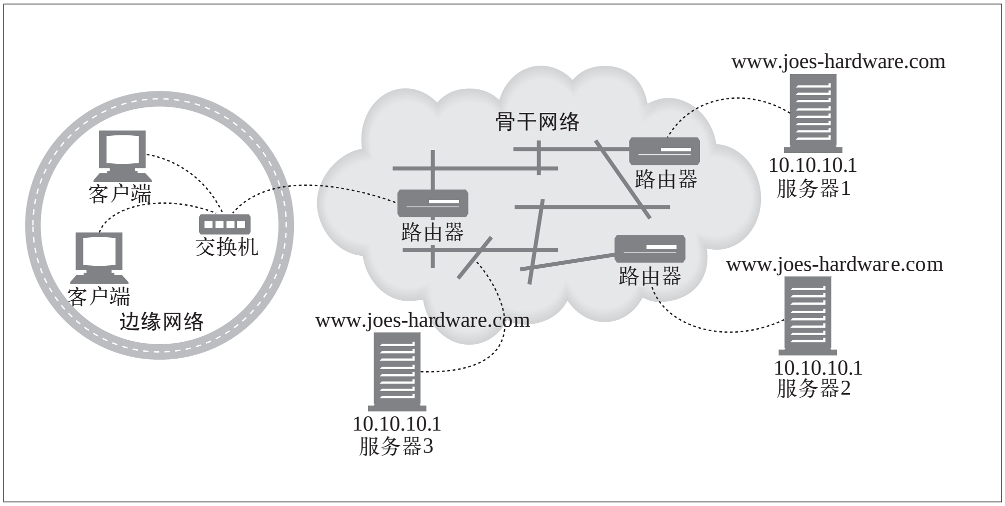


图 20-5　分布式任播寻址 458

20.4.4　IP MAC转发

在以太网中，HTTP 报文都是以携带地址的数据分组的形式发送的。每个分组都有一个第四层地址，由源 IP 地址、目的 IP 地址以及 TCP 端口号组成，它是第四层设备所关注的地址。每个分组还有一个第二层地址，MAC（Media Access Control，媒体访问控制）地址，这是第二层设备（通常是交换机和 Hub）所关注的地址。第二层设备的任务是接收具有特定输入 MAC 地址的分组，然后将其转发到特定的输出 MAC 地址上去。

比如，图 20-6 中交换机的程序会将来自 MAC 地址 MAC3 的所有流量都发送到 MAC 地址 MAC4 上去。

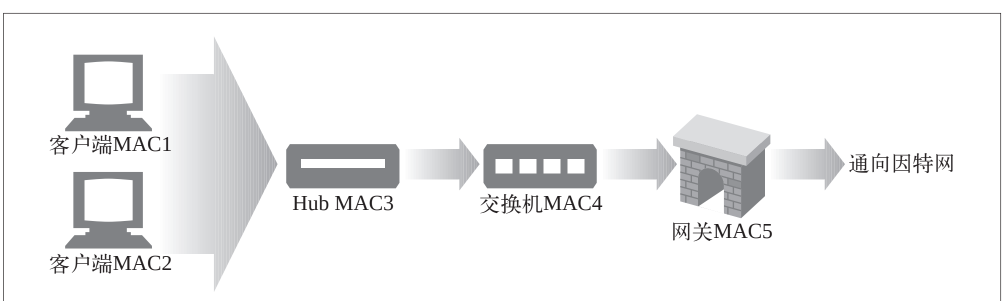


图 20-6　第二层交换机会将客户端请求发送给一个网关

第四层交换机能够检测出第四层地址（IP 地址和 TCP 端口号），并据此来选择路由。比如，一台第四层交换机可以将所有目的为端口 80 的 Web 流量都发送到某个代理上去。在图 20-7 中，编写交换机程序，将 MAC3 上所有端口 80 的流量都转发到 MAC6（代理缓存）上去。MAC3 上所有其他流量都会被转发到 MAC5 上去。

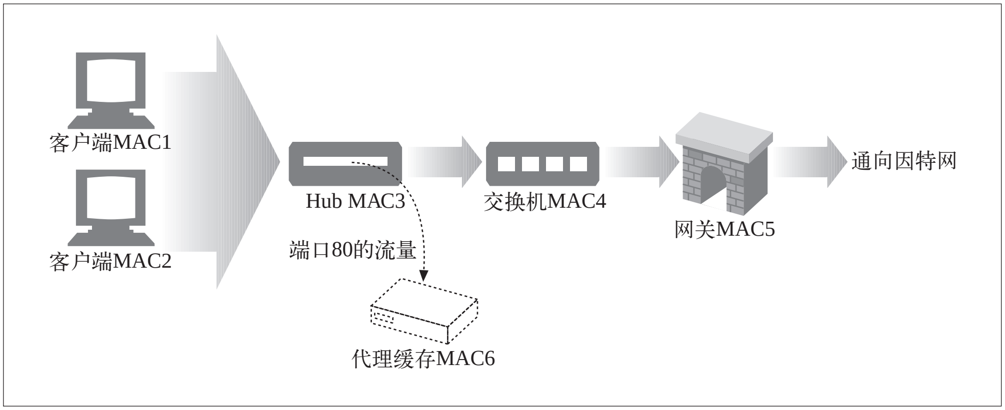


图 20-7　通过第四层交换机进行 MAC 转发

通常，如果缓存中有所请求的 HTTP 内容，而且是新鲜的，那么就由代理缓存来提供内容；否则，代理缓存就会代表客户端向此内容的原始服务器发送一条 HTTP 请求。交换机会将端口 80 的请求从代理（MAC6）发送给因特网网关（MAC5）。

支持 MAC 转发的第四层交换机通常会将请求转发给几个代理缓存，并在它们之间
459 平衡负载。类似地，也可以将 HTTP 流量转发给备用 HTTP 服务器。

因为 MAC 地址转发只是点对点的，所以服务器或代理只能位于离交换机一跳远的地方。

20.4.5　IP地址转发

在 IP 地址转发中，交换机或其他第四层设备会检测输入分组中的 TCP/IP 地址，并通过修改目的 IP 地址（不是目的 MAC 地址），对分组进行相应的转发。与 MAC 转发相比，这么做的优点是目标服务器不需要位于一跳远的地方；只需要位于交换机的上游就行了，而且通常第三层的端到端因特网路由都会将分组传送到正确的地方。这种类型的转发也被称为 NAT（Network Address Translation，网络地址转换）。

但还有一个问题，就是对称路由。从客户端接受输入 TCP 连接的交换机管理着连接；交换机必须通过那条 TCP 连接将响应回送给客户端。这样，所有来自目标服务器或代理的响应都必须返回给交换机（参见图 20-8）。

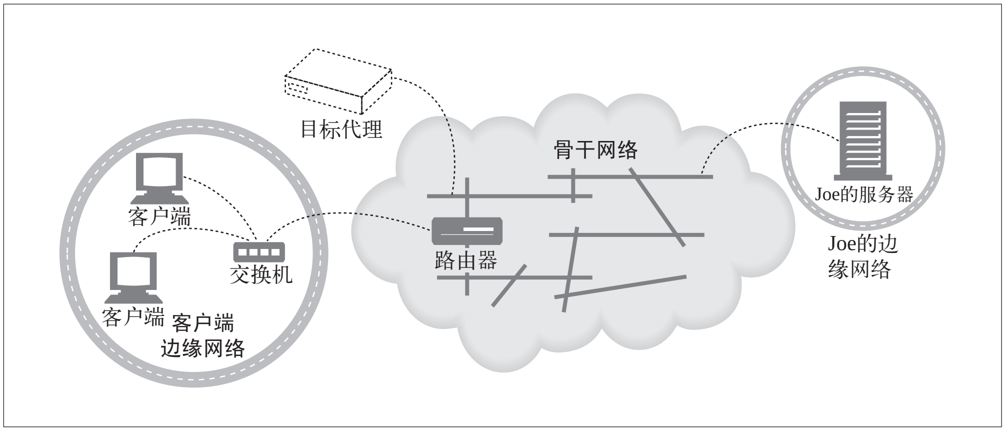


图 20-8　对代理缓存或镜像 Web 服务器进行 IP 地址转发的交换机　460

有以下两种方式可以控制响应的返回路径。

- 将分组的源 IP 地址改成交换机的 IP 地址。通过这种方式，无论交换机和服务器之间采用何种网络配置，响应分组都会被发送给交换机。这种方式被称为完全 NAT（full NAT），其中的 IP 转发设备会对目的 IP 地址和源 IP 地址都进行转换。图 20-9 显示了对一条 TCP/IP 数据报进行完全 NAT 的效果。这样做的缺点是服务器不知道客户端的 IP 地址，那种需要认证和计费的 Web 服务器无法获知客户端的 IP 地址。
- 如果源 IP 地址仍然是客户端的 IP 地址，就要确保（从硬件的角度来看）没有从服务器到客户端的直接路由（绕过交换机的）。这种方式有时被称为半 NAT（half NAT）。这种方法的优点是服务器知道客户端的 IP 地址，但缺点是要对客户端和服务器之间的整个网络都有某种程度的控制。

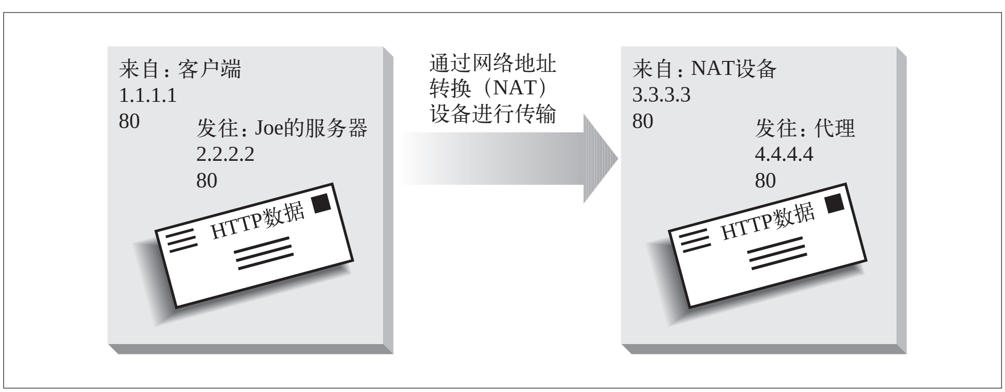


图 20-9　一条 TCP/IP 数据报的完全 NAT

20.4.6 网元控制协议

NECP（Network Element Control Protocol，网元控制协议）允许网元（NE，路由器和交换机等负责转发 IP 分组的设备）与服务器元素（SE，Web 服务器和代理缓存等提供应用层请求的设备）进行交互。NECP 并未显式提供对负载均衡的支持；它只是为 SE 提供了一种发送负载均衡信息给 NE 的方式，这样 NE 就可以在它认为合适的情况下进行负载均衡了。与 WCCP 一样，NECP 也提供了几种转发分组的方式：MAC 转发、GRE 封装和 NAT。

NECP 支持例外。SE 可以决定它不能为某些特定的源 IP 地址提供服务，并将这些地址发送给 NE。然后，NE 可以将来自这些 IP 地址的请求转发给原始服务器。

报文

461 表 20-3 描述了 NECP 报文。

表20-3　NECP报文

报　　文	发　送　者	含　　义
NECP_NOOP		不操作——什么都不做
NECP_INIT	SE	SE 初始化了与 NE 的通信。SE 打开与 NE 的 TCP 连接后，将此报文发送给它。SE 必须知道要连接哪个 NE 端口
NECP_INIT_ACK	NE	确认 NECP_INIT
NECP_KEEPALIVE	NE 或 SE	询问对等实体是否仍然活跃
NECP_KEEPALIVE_ACK	NE 或 SE	对 keep-alive 报文的应答
NECP_START	SE	SE 说："我在这里，做好接受网络流量的准备了。"可以指定端口
NECP_START_ACK	NE	确认 NECP_START
NECP_STOP	SE	SE 告诉 NE"停止向我发送流量。"
NECP_STOP_ACK	NE	NE 确认 NECP_STOP 操作
NECP_EXCEPTION_ADD	SE	SE 说要向 NE 列表中添加一个或多个例外。例外可以基于源 IP 地址、目的 IP 地址、（IP 之上的）协议，或者端口
NECP_EXCEPTION_ADD_ACK	NE	对 NECP_EXCEPTION_ADD 进行证实
NECP_EXCEPTION_DEL	SE	请求 NE 从列表中删除一个或多个例外
NECP_EXCEPTION_DEL_ACK	NE	对 NECP_EXCEPTION_DEL 进行证实
NECP_EXCEPTION_RESET	SE	请求 NE 删除整个例外列表
NECP_EXCEPTION_RESET_ACK	NE	对 NECP_EXCEPTION_RESET 进行证实
NECP_EXCEPTION_QUERY	SE	查询 NE 的整个例外列表
NECP_EXCEPTION_RESP	NE	响应例外查询

20.5　代理的重定向方法

到目前为止，我们已经讨论过通用的重定向方法了。（出于潜在的安全考虑）内容也可能需要通过各种代理来访问，或者网络中可能有一个客户端可利用的代理缓存。（因为获取已缓存的内容很可能要比直接连接到原始服务器快得多。）

但 Web 浏览器客户端怎么才会知道要连接到某个代理上去呢？可以用 3 种方法来判断：显式的浏览器配置，动态自动配置以及透明拦截。我们会在本节中讨论这 3 种技术。

代理可以顺次将客户端请求重定向到另一个代理上去。比如，没有缓存此内容的代理缓存可能会选择将客户端重定向到另一个代理缓存。这样一来，响应就会来自与客户端请求资源的地址不同的另外一个地址，所以，我们还会讨论几种用于对等代理 – 缓存重定向的协议：ICP、CARP 和 HTCP。 462

20.5.1　显式浏览器配置

大多数浏览器都可以配置为从代理服务器上获取内容——浏览器中有一个下拉菜单，用户可以在这个菜单中输入代理的名字或 IP 地址以及端口号。然后浏览器的所有请求都可以发送给这个代理。有些服务提供商不允许用户配置普通浏览器来使用代理，它们会要求用户下载事先配置好的浏览器。这些浏览器知道所要使用的代理的地址。

显式浏览器配置有以下两个主要的缺点。

- 配置为使用代理的浏览器，即使在代理无法响应的情况下，也不会去联系原始服务器。如果代理崩溃了，或者没有正确配置浏览器，用户就会遇到连接方面的问题。
- 对网络架构进行修改，并将这些修改通知给所有的终端用户都是很困难的。如果服务提供商要添加更多的代理服务器，或者使其中一些退出服务，用户都要修改浏览器代理设置。

20.5.2　代理自动配置

显式配置浏览器使其联系特定的代理，这样会限制网络架构方面的变动，因为它是靠用户来介入并重新配置浏览器的。自动配置方式可以动态配置浏览器，连接到正确的代理服务器，以解决这个问题。这种方法已经实现了，被称为代理自动配置（PAC）协议。PAC 是网景公司定义的，网景公司的 Navigator 和微软的 Inernet Explorer 浏览器都支持此协议。

PAC 的基本思想是让浏览器去获取一个称为 PAC 的特殊文件，这个文件说明了每个 URL 所关联的代理。必须配置浏览器，为这个 PAC 文件关联一个特定的服务器。这样，浏览器每次重启的时候都可以获取这个 PAC 文件了。

PAC 文件是个 JavaScript 文件，其中必须定义函数：

```
function FindProxyForURL(url, host)
```

如下所示，浏览器要为请求的每条 URL 调用这个函数：

```
return_value = FindProxyForURL(url_of_request, host_in_url);
```

其返回值为一个字符串，用来说明浏览器应该到哪里请求这个 URL。返回值可以是所关联的代理名称列表（比如，`PROXY proxy1.domain.com`，`PROXY proxy2.domain.com`），或者是字符串 `"DIRECT"`，这个字符串说明浏览器应该绕开所有的代理，直接连接原始服务器。

463 图 20-10 给出了浏览器对 PAC 文件的请求以及响应此请求的操作顺序。在本例中，服务器回送了带有 JavaScript 程序的 PAC 文件。JavaScript 程序中有一个 `FindProxyForURL` 函数，用来告知浏览器，如果所请求的 URL 的主机位于 netscape.com 域中，就直接与原始服务器联系，所有其他请求都连接到 proxy1.joes-cache.com。浏览器会为它所请求的每个 URL 调用这个函数，并根据此函数返回的结果进行连接。

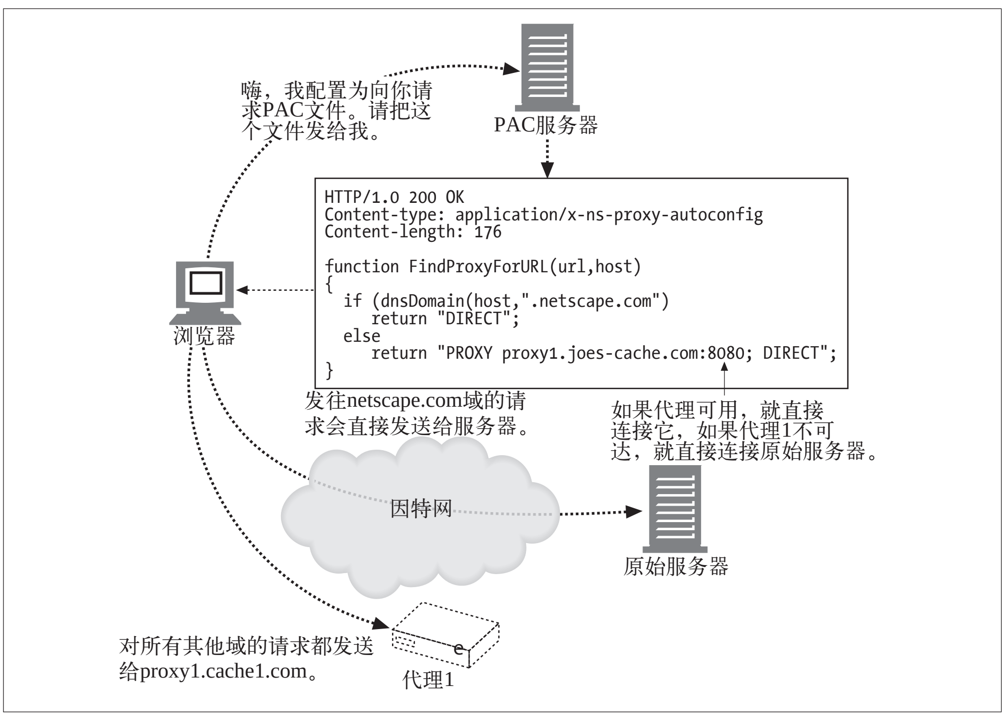


图 20-10　代理自动配置

PAC 协议是相当强大的：JavaScript 程序可以请求浏览器根据大量与主机名相关的参数来选择代理，比如 DNS 地址和子网，甚至星期几或具体时间。只要服务器中的 PAC 文件保持更新，能反映代理位置的变化，PAC 就允许浏览器根据网络结构的变化自动与合适的代理进行联系。PAC 存在的主要问题是必须要对浏览器进行配置，让它知道要从哪个服务器获取 PAC 文件，因此它就是一个全自动配置的系统。下一节讨论的 WPAD 解决了这个问题。

就像那些预配置浏览器一样，现在一些主要的 ISP 都在使用 PAC。

20.5.3　Web代理自动发现协议

WPAD（Web 代理自动发现协议）的目标是在不要求终端用户手工配置代理设置，而且不依赖透明流量拦截的情况下，为 Web 浏览器提供一种发现并使用附近代理的 464
方式。由于可供选择的发现协议有很多，而且不同浏览器的代理使用配置也存在差异，因此定义 Web 代理自动发现协议时，普通的问题会被复杂化。

本节包含了一个经过缩略，且重新组织过的 WPAD 因特网草案版本。现在，这个草案是作为 IETF 的 Web 中间人工作组的一部分开发的。

1. PAC文件自动发现

WPAD 允许 HTTP 客户端定位一个 PAC 文件，并使用这个 PAC 文件找到适当的代理服务器的名字。WPAD 不能直接确定代理服务器的名字，因为这样就无法使用 PAC 文件提供的附加功能了（负载均衡，请求路由到一组服务器上去，故障时自动转移到备用代理服务器等）。

如图 20-11 所示，WPAD 协议发现了 PAC 文件 URL，这个 URL 也被称为*配置* URL（CURL）。PAC 文件执行了一个 JavaScript 程序，这个程序会返回合适的代理服务器地址。

实现 WPAD 协议的 HTTP 客户端：

- 用 WPAD 找到 PAC 文件的 CURL；
- 根据这个 CURL 获取 PAC 文件（又名配置文件或 CFILE）；
- 执行 PAC 文件来确定代理服务器；
- 向 PAC 文件返回的那个代理服务器发送 HTTP 请求。

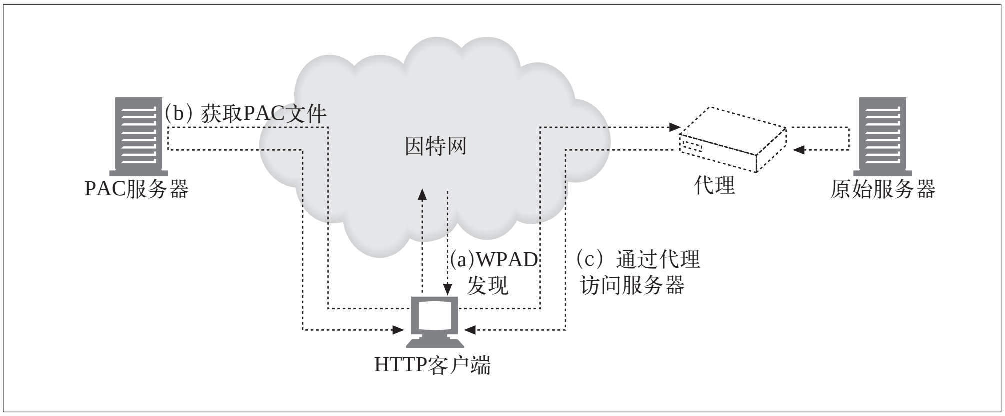


图 20-11　WPAD 确定了 PAC URL，这个 PAC 文件确定了代理服务器

2. WPAD算法

WPAD 使用了一系列资源发现技术来确定适当的 PAC 文件 CURL。并不是所有的
465 组织都可以使用所有技术的，所以 WPAD 指定了多种发现技术。在成功获得 CURL 之前，WPAD 客户端会一个个地尝试每种技术。

当前的 WPAD 规范按序定义了下列技术：

- DHCP（动态主机配置协议）；
- SLP（服务定位协议）；
- DNS 知名主机名；
- DNS SRV 记录；
- DNS TXT 记录中提供的服务 URL。

在这 5 种机制中，要求 WPAD 客户端必须支持 DHCP 和 DNS 知名主机名技术。我们会在后继小节中提供更多的细节。

WPAD 客户端会按顺序用上面提供的发现机制发送一系列资源发现请求。客户端只会尝试它们所支持的机制。只要某次发现尝试成功了，客户端就会用得到的信息来构建 PAC CURL。

如果从那个 CURL 上成功获取到 PAC 文件，这个过程就结束了。如果没有，客户端就从它在预定义的资源发现请求系列里中断的地方开始恢复。如果尝试了所有的发现机制之后，都没有获取到 PAC 文件，WPAD 协议就失败了，客户端会配置为不使用代理服务器。

客户端首先会尝试 DHCP，然后是 SLP。如果没有获取到 PAC 文件，客户端会继续

执行那些基于 DNS 的机制。

客户端会在 DNS SRV、知名主机名和 DNS TXT 记录等方法中循环多次。每次都使 DNS 查询的 QNAME 变得越来越不具体。通过这种方式，客户端就可以定位出尽可能具体的配置信息，但也可能会转而使用一些不太具体的信息。每次 DNS 查找都会在 QNAME 前加上 wpad，用以说明请求的资源类型。

考虑主机名为 johns-desktop.development.foo.com 的客户端。下面是一个完整的 WPAD 客户端会执行的发现尝试顺序：

- DHCP；
- SLP；
- 用 QNAME=wpad.development.foo.com 进行 DNS A 查找；
- 用 QNAME=wpad.development.foo.com 进行 DNS SRV 查找；
- 用 QNAME=wpad.development.foo.com 进行 DNS TXT 查找；
- 用 QNAME=wpad.foo.com 进行 DNS A 查找；
- 用 QNAME=wpad.foo.com 进行 DNS SRV 查找；
- 用 QNAME=wpad.foo.com 进行 DNS TXT 查找。 466

说明整个操作过程的详细伪代码请参见 WPAD 规范。后面的小节将讨论两种必备机制——DHCP 和 DNS A 查找。有关 CURL 发现方法的其他详细内容参见 WPAD 规范。

3. 用DHCP进行CURL发现

要使用这种机制，就必须将 CURL 存储在 WPAD 客户端可以查询的 DHCP 服务器上。WPAD 客户端可以通过向 DHCP 服务器发送 DHCP 查询来获取 CURL。（如果 DHCP 服务器中配置了这种信息，）就可以在 DHCP 可选代码 252 中获取 CURL。所有 WPAD 客户端实现都必须支持 DHCP。RFC 2131 详细介绍了 DHCP 协议。现存的 DHCP 选项列表参见 RFC 2132。

如果 WPAD 客户端已经在其初始化过程中执行了 DHCP 查询，DHCP 服务器可能就已经提供了那个值。如果无法通过客户端 OS API 获得这个值，客户端就向 DHCP 服务器发送一条 DHCPINFORM 报文，以获取这个值。

WPAD 的 DHCP 可选代码 252 为 STRING 类型，可以是任意长度。这个字符串中包含了一个指向适当 PAC 文件的 URL。比如：

```
"http://server.domain/proxyconfig.pac"
```

4. DNS A记录查找

要让这种机制工作，就必须将合适的代理服务器的 IP 地址存储在 WPAD 客户端可以查询的 DNS 服务器上。WPAD 客户端会向 DNS 服务器发送一个 A 记录查询，以获取 CURL。成功查询的结果中会包含合适的代理服务器的 IP 地址。

WPAD 客户端实现必须支持这种机制。这应该是很简单的，因为它只要求基本的 DNS A 记录查找。用知名 DNS 别名进行资源发现的详细过程请参见 RFC 2219。对 WPAD 来说，规范使用了“wpad”的“知名别名”来进行 Web 代理自动发现。

客户端执行了下列 DNS 查找：

```
QNAME=wpad.TGTDOM., QCLASS=IN, QTYPE=A
```

成功的查找中包含了 IP 地址，WPAD 客户端根据这个地址构建 CURL。

5. 获取PAC文件

只要创建了候选的 CURL，WPAD 客户端通常都会向 CURL 发送一条 GET 请求。发出请求时，WPAD 客户端必须要发送一些带有适当 CFILE 格式信息的 `Accept` 首
467 部，这些 CFILE 格式都是它们所能处理的。比如：

```
Accept: application/x-ns-proxy-autoconfig
```

而且，如果 CURL 的结果是要进行重定向，客户端就必须跟随这些重定向到其最终目的地。

6. 何时执行WPAD

至少要在出现以下情况的时候进行 Web 代理自动发现。

- 在 Web 客户端启动的时候——WPAD 只在第一个实例启动的时候执行。后面的实例会继承这种设置。
- 只要有来自网络栈的通知，就说明客户端主机的 IP 地址改变了。

哪个选项在其环境中有意义，Web 客户端就可以选择哪个。而且，客户端还必须根据 HTTP 的过期时间，为之前下载的 PAC 文件的过期时间尝试一个发现周期。PAC 文件过期时，客户端遵循过期时间，重新运行 WPAD 过程是很重要的。

如果 PAC 文件没有提供替换方案，在当前配置的代理失效的情况下，客户端还可以选择重新运行 WPAD 过程。

只要客户端决定使当前的 PAC 文件失效，就必须重新运行整个 WPAD 协议，以确

保它会发现当前正确的 CURL。具体来说，就是协议不能有条件地获取 PAC 文件的 `If-Modified-Since`。

WPAD 协议广播与 / 或多播通信可能需要大量的网络环回时间。WPAD 协议的激活频率不应该高于上面指定的频率（比如在每次获取 URL 时进行一次）。

7. WPAD欺骗

WPAD 的 IE 5 实现允许 Web 客户端在没有用户干预的情况下，自动检测代理设置。WPAD 使用的算法会在全称域名前加上主机名“wpad”，并会逐渐删除子域名，直到它找到能够响应主机名的 WPAD 服务器，或到达第三级域名。比如，域 a.b.microsoft.com 中的 Web 客户端会先查询 wpad.a.b.microsoft、wpad.b.microsoft.com，然后再查询 wpad.microsoft.com。

这样会暴露出一个安全漏洞，因为在国际应用（及其他特定的配置）中，第三级域名可能是不可信的。恶意用户可以建立一个 WPAD 服务器，并提供他选中的代理配置命令。后继（5.01 及以后）的 IE 版本修正了这个问题。

8. 超时

WPAD 会经过多个级别的发现，客户端必须确保每个阶段都有时限保证。可能的情况下，将每个阶段都限制在 10 秒以内是比较合理的，但实现者可能会选择其他更适 468
合其网络特性的值。比如，运行在无线网络上的设备实现，由于带宽较低或时延较长，可能就会使用更大的时限。

9. 管理者的考虑

管理者至少应该在其环境中配置 DHCP 或 DNS A 记录查找方式中的一种，因为只有这两种方式是所有兼容客户端都必须实现的。除此之外，通过配置环境使其支持搜索列表中顺序靠前的机制，可以缩短客户端的启动时间。

使用这种协议结构的主要动力之一是支持客户端定位附近的代理服务器。在很多环境中，都会有多个代理服务器（工作组、公司网关、ISP、骨干网等）。

在 WPAD 框架结构中，可以在很多地方确定代理服务器是否“邻近”。

- 不同子网的 DHCP 服务器会返回不同的答案。还可以根据客户端的 `cipaddr` 字段或客户端标识符选项作出决定。
- 可以对 DNS 服务器进行配置，使其为不同的域名后缀（比如，QNAME wpad.marketing.bigcorp.com 和 wpad.development.bigcorp.com）返回不同的 SRV/A/TXT 资源记录（RR）。

- 处理 CURL 请求的 Web 服务器会根据 `User-Agent` 首部、`Accept` 首部、客户端 IP 地址 / 子网 / 主机名、附近代理服务器的拓扑分布等作出决定。可能由处理 CURL 的 CGI 可执行文件进行这种处理。如前所述，甚至可能是某个处理 CURL 请求的代理服务器来作出这些决定。
- PAC 文件的表达能力可能足以在客户端运行时从一组候选的代理服务器中进行选择。CARP 就是在此基础上实现缓存阵列的。PAC 文件可以计算出到一组候选代理服务器的网络距离（或其他合理的度量方式），并选择"最近"或"响应最积极"的服务器，这并不是什么不可思议的事情。

20.6　缓存重定向方法

我们已经讨论过一些将流量重定向到通用服务器的技术，以及一些将流量导向代理或网关的专用技术了。这一节会介绍一些更复杂的、用于缓存代理服务器的重定向技术。这些技术要尽量做到可靠、高效且能感知内容——这样可以将请求分配到可
469 能包含特定内容的位置上去，因此比前面讨论过的那些协议更复杂。

WCCP重定向

Cisco 系统公司开发的 WCCP 可以使路由器将 Web 流量重定向到代理缓存中去。WCCP 负责路由器和缓存服务器之间的通信，这样路由器就可以对缓存进行验证（确保它们已启动且正在运行），在缓存之间进行负载均衡，并将特定类型的流量发送给特定的缓存了。WCCP 版本 2（WCCP2）是个开放的协议。这里我们会探讨 WCCP2。

1. WCCP重定向是怎样工作的

下面是 WCCP 重定向在 HTTP 上工作过程的概述（WCCP 对其他协议的重定向过程也是类似的）。

- 启动包含了一些支持 WCCP 的路由器和缓存的网络，这些路由器和缓存之间可以相互通信。
- 一组路由器及其目标缓存构成一个 WCCP 服务组。服务组的配置说明了要将何种流量发往何处、流量是如何发送的以及如何在服务组的缓存之间进行负载均衡。
- 如果服务组配置为重定向 HTTP 流量，服务组中的路由器就会将 HTTP 请求发送给服务组中的缓存。
- HTTP 请求抵达服务组中的路由器时，路由器会（根据对请求 IP 地址的散列，或者"掩码 / 值"的配对策略）选择服务组中的某个缓存为请求提供服务。

- 路由器向缓存发送请求分组，可以用缓存的 IP 地址来封装分组，也可以通过 IP MAC 转发来实现。
- 如果缓存无法为请求提供服务，就将分组返回给路由器进行普通的转发。
- 服务组中的成员会互相交换心跳报文，不断验证对方的可用性。

2. WCCP2报文

WCCP2 报文有 4 种，如表 20-4 所示。

表20-4　WCCP2报文

报　文　名	发　送　者	所承载的信息
WCCP2_HERE_I_AM	从缓存发送给路由器	这些报文告诉路由器，缓存可以接收流量。报文中包含了该缓存的服务组的所有信息。只要有缓存加入了服务组，它就会将这些报文发送给组里所有的路由器。可以通过这些报文与发送 WCCP2_I_SEE_YOU 报文的路由器进行沟通
WCCP2_I_SEE_YOU	从路由器发送给缓存	这些报文是对 WCCP2_HERE_I_AM 报文的响应。可以通过这些报文对分组转发方式、分配方法（哪个是指定的缓存）、分组返回方法和安全性进行沟通 470
WCCP2_REDIRECT_ASSIGN	从指定的缓存发送给路由器	这些报文为负载均衡分配任务，它们可以为散列表负载均衡发送桶信息，或为“掩码 / 值”负载均衡发送“掩码 / 值”组对信息
WCCP2_REMOVAL_QUERY	从路由器发送给在 2.5× HERE_I_AM_T 秒内没有发送过 WCCP2_HERE_I_AM 报文的缓存	如果路由器没有周期性地收到 WCCP2_HERE_I_AM 报文，路由器就会发送此报文，以查看是否应该将这个缓存从服务组中删除掉。来自缓存的正确响应为三条完全相同的 WCCP2_HERE_I_AM 报文，中间间隔 HERE_I_AM_T/10 秒

WCCP2_HERE_I_AM 的报文格式为：

```
WCCP Message Header
Security Info Component
Service Info Component
Web-cache Identity Info Component
Web-cache View Info Component
Capability Info Component （可选）
Command Extension Component （可选）
```

WCCP2_I_SEE_YOU 的报文格式为：

```
WCCP Message Header
Security Info Component
```

```
Service Info Component
Router Identity Info Component
Router View Info Component
Capability Info Component （可选）
Command Extension Component （可选）
```

WCCP2_REDIRECT_ASSIGN 的报文格式为：

```
WCCP Message Header
Security Info Component
Service Info Component
Assignment Info Component, or Alternate Assignment Component
```

WCCP2_REMOVAL_QUERY 的报文格式为：

```
WCCP Message Header
Security Info Component
Service Info Component
Router Query Info Component
```

3. 报文组件

每条 WCCP2 报文都由一个首部和一些组件构成。WCCP 首部信息包含报文类型（Here I Am、I See You、Assignment 或 Removal Query）、WCCP 版本和报文长度（不包括首部的长度）。

每个组件都以一个描述组件类型和长度的 4 字节首部开始。组件长度不包括组件首
471 部的长度。报文组件如表 20-5 所述。

表20-5　WCCP2报文组件

组　　件	描　　述
安全信息	包含安全选项和安全实现。安全选项可以是： WCCP2_NO_SECURITY (0) WCCP2_MD5_SECURITY (1) 如果选项是不安全的（NO），安全实现字段就不存在。如果选项为 MD5，安全实现字段就是一个包含了报文校验和、服务组密码的 16 字节的字段。密码不能超过 8 个字节
服务信息	描述服务组。服务类型 ID 可以使用两个值： WCCP2_SERVICE_STANDARD (0) WCCP2_SERVICE_DYNAMIC (1) 如果服务类型是标准的（STANDARD），服务就是完全由服务 ID 定义的知名服务。HTTP 就是个知名服务。如果服务类型是动态的（DYNAMIC），就由下列设置来定义服务：优先级、协议、服务标记（决定是否散列）以及端口
路由器身份信息	包含路由器的 IP 地址和 ID，并（通过 IP 地址）列出了路由器想要通信的所有 Web 缓存

（续）

组　　件	描　　述
Web 缓存身份信息	包含了 Web 高速缓存的 IP 地址和重定向散列表映射
路由器视图信息	包含了路由器的服务组视图（路由器和缓存的身份）
Web 缓存视图信息	包含了 Web 缓存的服务组视图
分配信息	显示了如何将 Web 缓存分配到特定的散列桶中去
路由器查询信息	包含了路由器的 IP 地址、所要查询的 Web 缓存的地址以及服务组中最近从 Web 缓存中收到 Here I Am 报文的路由器 ID
能力信息	那些需要广告所支持的分组转发方式、负载均衡和分组返回方式的路由器会使用此信息。那些要让路由器知道它希望使用哪种方法的 Web 缓存也会使用此信息
替换分配	包含了负载均衡的散列表分配信息
分配图	包含了服务组的“掩码 / 值”设置元素
命令扩展	Web 缓存用它来告诉路由器它们正在关机。路由器用它来确认缓存是否关闭

4. 服务组

服务组（service group）由一组支持 WCCP 的路由器和缓存组成，它们之间可以交换 WCCP 报文。路由器会向服务组中的缓存发送 Web 流量。服务组的配置确定了如何将流量分配到服务组的缓存中去。路由器和缓存会在 Here I Am 和 I See You 报文中交换服务组的配置信息。

5. GRE分组封装

支持 WCCP 的路由器会用服务器的 IP 地址将 HTTP 分组封装起来，将其重定向到特定的服务器上去。分组封装中还包含了 `IP` 首部的 `proto` 字段，用来说明通用路由器封装（GRE）。`proto` 字段的存在告诉接收代理，它有一个封装的分组。分组被封装起来，客户端的 IP 地址就不会丢失了。图 20-12 显示了 GRE 分组的封装过程。 472

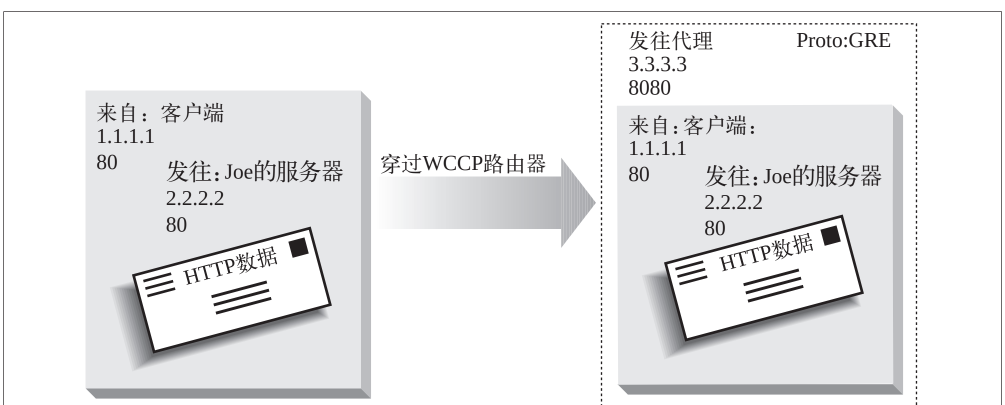


图 20-12　WCCP 路由器如何修改 HTTP 分组的目的 IP 地址

6. WCCP的负载均衡

除了路由功能之外，WCCP 路由器还可以在几个接收服务器之间进行负载均衡。WCCP 路由器及其接收服务器会交换*心跳报文*（heartbeat message），以便相互通知自己处于启动运行状态。如果某特定接收服务器停止发送心跳报文，WCCP 路由器就会将请求流量直接发送到因特网上，而不会将其重定向给那个节点。节点重新提供服务时，WCCP 路由器会再次开始接收心跳报文，并继续向节点发送请求流量。

20.7 因特网缓存协议

ICP（因特网缓存协议）允许缓存在其兄弟缓存中查找命中内容。如果某个缓存中没有 HTTP 报文所请求的内容，它可以查明内容是否在附近的兄弟缓存中，如果在，就从那里获取内容，以避免查询原始服务器而带来的更多开销。可以把 ICP 当作一个缓存集群协议。HTTP 请求报文的最终目的地可以通过一系列的 ICP 查询确定，从这个角度来说，它就是一个重定向协议。

ICP 是一个对象发现协议。它会同时去询问附近的多个缓存，看看它们的缓存中是否有特定的 URL。附近的缓存如果有那个 URL 的话，就会返回一个简短的报文 HIT，如果没有，就返回 MISS。然后，缓存就可以打开一条到拥有此对象的邻居缓存的 HTTP 连接了。

ICP 是很简单直接的。ICP 报文是一个以网络字节序表示的 32 位封装结构，这样更
473 便于进行解析。为了提高效率，可以由 UDP 数据报承载其报文。UDP 是一种不可靠的因特网协议，说明在传输的过程中数据可能会被破坏，因此使用 ICP 的程序要具有超时功能，以检测丢失的数据报。

下面简要描述一下 ICP 报文中的部分信息。

- Opcode（*操作码*）
  Opcode 是个 8 位的二进制值，用以描述 ICP 报文的含义。基本的 opcode 包括 ICP_OP_QUERY 请求报文和 ICP_OP_HIT 和 ICP_OP_MISS 响应报文。

- *版本*
  8 位的版本号描述了 ICP 协议的版本编号。Squid 使用的 ICP 版本记录在 RFC 2186 第 2 版中。

- *报文长度*
  以字节为单位的 ICP 报文总长。因为只有 16 位，所以 ICP 报文的长度不能超过 16 383 字节。URL 通常都小于 16 KB，如果超过这个长度，很多 Web 应用程序就无法处理它了。

- *请求编号*

  支持 ICP 的缓存会用请求编号来记录多个同时发起的请求和响应。ICP 应答报文数必须与触发应答的 ICP 请求报文数相同。

- *选项*

  32 位的 ICP 选项字段是个包含了若干标记的位矢量，这些标记可用来修改 ICP 的行为。ICPv2 定义了两个标记，这两个标记都会修改 ICP_OP_QUERY 请求。ICP_FLAG_HIT_OBJ 标记用来启动或禁止在 ICP 响应中返回文档数据。ICP_FLAG_SRC_RTT 标记请求由兄弟缓存测量的、到原始服务器的环回时间的估计值。

- *可选数据*

  保留了 32 位的可选数据用于可选特性。ICPv2 使用了可选数据的低 16 位来装载从兄弟缓存到原始服务器的可选环回时间的估计值。

- *发送端主机地址*

  承载了报文发送端 32 位 IP 地址的著名字段。实际中并未使用。

- *净荷*

  净荷内容的变化取决于报文的类型。对 ICP_OP_QUERY 来说，净荷是一个 4 字节的原始请求端主机地址，后面跟着一个由 NUL 结尾的 URL。对 ICP_OP_HIT_OBJ 来说，净荷是一个由 NUL 结尾的 URL，后面跟着一个 16 位的对象长度，接着是对象数据。

  更多有关 ICP 的信息，请参见 RFC 2186 和 RFC 2187。从美国应用网络研究国家实验室的网站上（http://www.nlanr.net/Squid/）也可以获得一些很棒的有关 ICP 和对等实体的参考文献。 474

20.8 缓存阵列路由协议

代理服务器通过拦截来自单个用户的请求，提供所请求 Web 对象的缓存副本，极大地降低了发往因特网的流量。但随着用户数的增加，大量流量可能会使代理服务器自身超载。

对此问题的一种解决方案就是使用多个代理服务器将负载分散到一组服务器上。CARP（缓存阵列路由协议）是微软公司和网景公司提出的一个标准，通过这个协议来管理一组代理服务器，使这组代理服务器对用户来说就像一个逻辑缓存一样。

CARP 是 ICP 的一个替代品。CARP 和 ICP 都允许管理者通过使用多个代理服务器来提高性能。本节讨论了 CARP 与 ICP 的区别，用 CARP 代替 ICP 的优缺点以及

CARP 协议实现上的一些技术细节。

ICP 中出现缓存未命中时，代理服务器会用 ICP 报文格式来查询附近的缓存，以确定 Web 对象是否存在。附近的缓存会以 HIT 或 MISS 进行响应，请求代理服务器会用这些响应来选择能够获取到对象的最适当的位置。如果 ICP 代理服务器是以层次结构排列的，未命中的查询会被提交给其父代理。图 20-13 以图形方式显示了如何通过 ICP 来解决命中和未命中的问题。

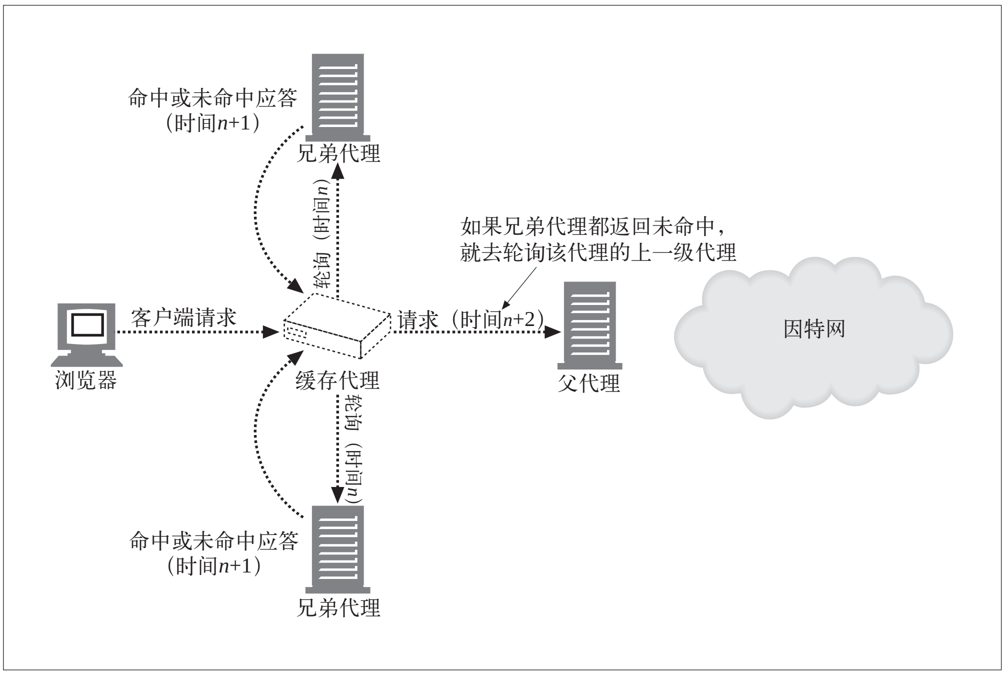


图 20-13　ICP 查询

475 注意，通过 ICP 协议连接起来的每个代理服务器都是将内容进行了冗余镜像的独立缓存服务器，这就说明在不同的代理服务器之间复制 Web 对象条目是可行的。相反，用 CARP 连接起来的一组服务器会被当作一个大型的服务器，其中每个组件服务器都只包含全部缓存文档中的一部分。通过对某个 Web 对象的 URL 应用散列函数，CARP 就可以将此对象映射到特定的代理服务器上去。每个 Web 对象都有一个唯一的家，所以我们可以通过单次查找确定对象的位置，而无须去查询集合中配置的每个代理服务器。图 20-14 总结了 CARP 重定向的方式。

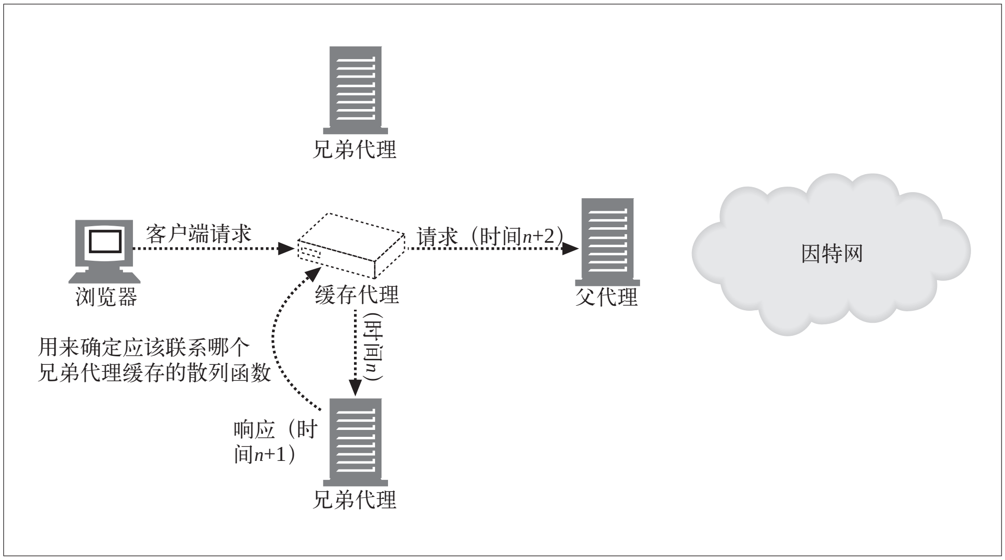


图 20-14　CARP 重定向

如图 20-14 所示，作为客户端和代理服务器中间人的缓存代理可以在各个代理服务器之间分配负载，但这项功能也可以由客户端自身提供。可以配置 Internet Explorer 和网景公司的 Navigator 这样的商用浏览器，以插件的形式计算散列函数，来确定应该把请求发送给哪个代理服务器。

CARP 对代理服务器做出的确定性解析说明它无须向所有邻居发送查询，这也就意味着这种方法所需发送的缓存间报文会比较少。随着越来越多的代理服务器添加到配置系统中来，缓存系统集群的规模会变得相当大。但 CARP 的一个缺点就是，如果某个代理服务器不可用了，就要重新修改散列表以反映这种变化，而且必须重新配置现存代理服务器上的内容。如果代理服务器经常崩溃的话，这么做的开销可能 476
会很高。相反，ICP 代理服务器中存在的冗余内容就表示它不需要重新配置。另一个潜在的问题是，由于 CARP 是个新协议，CARP 集群中可能不会包含那些现存的、只运行 ICP 协议的代理服务器。

介绍了 CARP 和 ICP 间的区别之后，我们来详述一下 CARP。CARP 重定向方法要完成下列任务。

- 保存一个参与 CARP 的代理服务器列表。周期性地查询这些代理服务器，看看它们是否仍然活跃。
- 为每个参与的代理服务器计算一个散列函数。散列函数的返回值要考虑此代理所能处理的负载量。

- 定义一个独立的散列函数,这个函数会根据所请求 Web 对象的 URL 返回一个数字。
- 将 URL 散列函数的结果代入代理服务器的散列函数，得到一个数字阵列。这些数字中的最大值决定了要为这个 URL 使用的代理服务器。由于算出来的值是确定的，所以对同一个 Web 对象的后继请求会被转发给同一台代理服务器。

以上 4 项任务可以由浏览器、插件执行，也可以在一个中间服务器上计算。

为每个代理服务器集群创建一个表，表中列出了集群中的所有服务器。表中的每个条目都应该包含全局参数的相关的信息，比如，负载因子、生存时间（TTL）、倒计数值和应该以何频率查询成员之类的全局参数。负载因子说明机器可以处理多少负载，这取决于那台机器的 CPU 速度和硬盘容量。可以通过 RPC 接口对此表进行远程维护。只要表中的字段被 RPC 修改了，就可以使其对下游的客户端和代理可见，或将其发布给它们。这项发布工作是在 HTTP 中进行的，这样，所有的客户端或代理服务器就都可以在不引入另一种代理间协议的基础上消化表格信息了。客户端和代理服务器只用了一个知名 URL 来获取这张表。

所使用的散列函数必须能够确保 Web 对象在参与的代理服务器间是统计分布的。应该用代理服务器的负载因子来确定分配给那台代理的 Web 对象的统计概率。

总之，CARP 协议允许将一组代理服务器看成单个的集群缓存，而不是（像 ICP 中那样的）一组相互合作但又相互独立的缓存服务器。确定的请求解析路径会在一跳
477 内找到某个特定的 Web 对象的家。这样会降低 ICP 在一组代理服务器中查找 Web 对象时常会产生的代理间流量。CARP 还可以避免在不同的代理服务器上存储 Web 对象的多个副本的问题，这样做的优点是缓存系统集群的 Web 对象存储容量较大，缺点是任意一个代理的故障都要改写现存代理的部分缓存内容。

20.9 超文本缓存协议

之前我们讨论了 ICP，这个协议允许代理缓存向兄弟缓存查询文件是否存在。但设计 ICP 时考虑的是 HTTP/0.9 协议，因此，向兄弟缓存查询资源是否存在时，只允许缓存发送 URL。HTTP 版本 1.0 和 1.1 引入了很多新的请求首部，这些首部可以和 URL 一起用来确定文件是否匹配。因此，只在请求中发送 URL 可能无法得到精确的响应。

HTCP（超文本缓存协议）允许兄弟缓存之间通过 URL 和所有的请求及响应首部来相互查询文档是否存在，以降低错误命中的可能。而且 HTCP 允许兄弟缓存监视或请求在对方的缓存中添加或删除所选中的文档，并修改对方已缓存文档的缓存策略。

图 20-13 说明了一个 ICP 事务，此图也可以用来说明 HTCP 事务，后者是另一个对象发现协议。如果附近的缓存中有这个文档，发起请求的缓存可以打开一条到此缓存的 HTTP 连接，以获取那个文档的副本。ICP 和 HTCP 事务之间的区别体现在请求和响应细节上。

HTCP 报文的结构如图 20-15 所示。首部中包含了报文的长度和报文版本。数据部分开始是数据长度，包含了 opcode、响应代码、一些标记及 ID，最后是实际的数据。可选的认证部分跟在 Data 小节的后面。

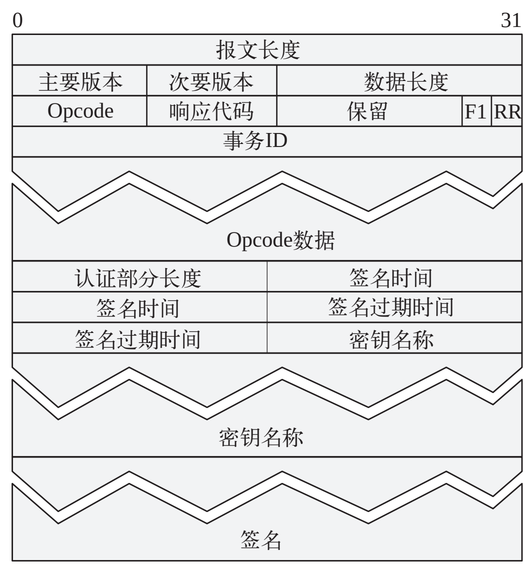


图 20-15　HTCP 报文格式

报文字段的详细内容如下所述。

- 首部

  Header 部分包含 32 位的报文长度，8 位的主要协议版本和 8 位的次要协议版本。报文长度包含所有首部、数据和认证部分的长度。

- 数据

  Data 部分包含了 HTCP 报文，结构如图 20-15 所示。数据组件如表 20-6 所示。 478

表20-6　HTCP数据组件

组　　件	描　　述
数据长度	16 位的 Data 部分字节数，包含 Length 字段自身的长度
Opcode	HTCP 事务的 4 位操作代码。表 20-7 列出了 Opcode 的完整内容
响应代码	说明事务成功或失败的 4 位键值。可能的值有： 0——没有进行认证，但需要进行认证； 1——需要进行认证，但没有得到满足； 2——未实现的 Opcode； 3——不支持主要版本； 4——不支持次要版本； 5——不合适、不允许或非预期的 Opcode。
F1	F1 是重载的——如果报文是一条请求，F1 就是请求端设置的 1 位标记，说明需要响应（F1=1）；如果报文是一条响应，F1 就是一个 1 位标记，用来说明应该将响应作为对整条报文的响应来解释（F1=1），还是将其作为对 Opcode 数据字段的响应来解释（F1=0）
RR	用来说明报文是请求（RR=0）还是响应（RR=1）的 1 位标记
事务 ID	32 位的值，与请求端的网络地址组合在一起可以唯一地标识 HTCP 事务
479 Opcode 数据	Opcode 数据与 Opcode 有关。参见表 20-7

表 20-7 列出了 HTCP Opcode 代码及其相应的数据类型。

表20-7　HTCP Opcode

Opcode	值	描　　述	响应代码	Opcode数据
NOP	0	本质上是一个 ping 操作	总是 0	无
TST	1		如果有实体，就为 0，如果没有提供实体，就为 1	在请求中包含 URL 和请求首部，在响应中只包含响应首部
MON	2		接受就为 0，拒绝就为 1	
SET	3	SET 报文允许缓存请求修改缓存策略。可以用于 SET 报文的首部参见表 20-9	接受就为 0，忽略就为 1	
CLR	4		如果曾经有过，但现在没有了，就为 0；如果曾经有过，而且现在还有，就为 1；如果从未有过，就为 2	

20.9.1　HTCP认证

HTCP 报文的认证部分是可选的。其结构如图 20-15 所示，表 20-8 列出了它的认证组件。

表20-8　HTCP认证组件

组　件	描　述
认证部分长度	16 位的报文认证部分字节数，包含了长度字段自身的长度
签名时间	32 位数，表示从格林尼治标准时间 1970 年 1 月 1 日 00:00:00 开始，到产生签名的时间之间的秒数
签名过期时间	32 位数，表示从格林尼治标准时间 1970 年 1 月 1 日 00:00:00 开始，到签名过期时所经历的秒数
密钥名称	用来表示共享密钥名称的字符串。密钥字段有两个部分：用来说明后面那个字符串长度的 16 位的字节数，后面跟着的字符串是未经解释的字节流
签名	HMAC-MD5 摘要，它是 B 值为 64（表示源 IP 地址和目的 IP 地址及端口）、报文的主要及次要 HTCP 版本、签名时间和签名过期值，完整的 HTCP 数据以及密钥的摘要。签名也包含两个部分：16 位长的字符串字节数，后面跟着这个字符串

20.9.2　设置缓存策略

SET 报文允许缓存请求对已缓存文档的缓存策略进行修改。表 20-9 中给出了可以在 SET 报文中使用的首部。 480

表20-9　修改缓存策略的缓存首部列表

首　部	描　述
`Cache-Vary`	请求端已经知道内容会随一组首部的变化而变化，这组首部与响应 `Vary` 首部中的那一组不同。这个首部会覆盖响应的 `Vary` 首部
`Cache-Location`	可能有此对象副本的代理缓存的列表
`Cache-Policy`	关于此对象的缓存策略，请求端已经了解到的比响应首部中指定的更详细。可能的值包括：`no-cache`，说明响应是不可缓存的，但可以在多个同时发起请求的请求端之间共享；`no-share`，说明对象是不可共享的；`no-cache-cookie`，说明内容可能会随 cookie 而发生变化，不推荐缓存
`Cache-Flags`	请求端修改了对象的缓存策略，可能要对它进行特别的处理，不一定要根据其实际的策略进行处理
`Cache-Expiry`	发送端了解到的文档实际过期时间
`Cache-MD5`	请求端计算出来的对象的 MD5 校验和，可能与 `Content-MD5` 首部的值有所不同，也可能在对象没有 `Content-MD5` 首部的情况下提供
`Cache-to-Origin`	请求端测量的到原始服务器的往返时间。此首部值的格式为 <原始服务器名称或 IP 地址><以秒为单位的平均往返时间><采样数><请求端和原始服务器之间的路由器跳数>

HTCP 允许通过查询报文将请求和响应首部发送给兄弟缓存，这样可以降低缓存查询中的错误命中率。通过进一步允许在兄弟缓存间交换策略信息，HTCP 还可以提高兄弟缓存之间的合作能力。

20.10 更多信息

更多信息可参考下列文献。

- *DNS and Bind*[3]（《DNS 与 BIND》）
  Cricket Liu、Paul Albitz 和 Mike Loukides 著，O'Reilly & Associates 公司出版。

- http://www.wrec.org/Drafts/draft-cooper-Webi-wpad-00.txt
  "Web Proxy Auto-Discovery Protocol"（"Web 代理自动发现协议"）。

- http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html
  "Navigator Proxy Auto-Config File Format"（"Navigator 代理自动配置文件格式"）。

- http://www.ietf.org/rfc/rfc2186.txt
  IETF RFC 2186，"Intercache Communication Protocol (ICP) Version2"（"缓存间的通信协议，版本 2"），D. Wessels 和 K. Claffy 编写。

- http://icp.ircache.net/carp.txt
  "Cache Array Routing Protocol v1.0"（"缓存阵列路由协议 v1.0"）。

- http://www.ietf.org/rfc/rfc2756.txt
  481 IETF RFC 2756，"Hyper Text Caching Protocol(HTCP/0.0)"（"超文本缓存协议（HTCP/0.0）"），P. Vixie 和 D. Wessels 编写。

- http://www.ietf.org/internet-drafts/draft-wilson-wrec-wccp-v2-00.txt
  "Web Cache Communication Protocol v2.0"（"Web 缓存通信协议 v2.0"），M. Cieslak、D. Forster、G. Tiwana 和 R. Wilson 编写。

- http://www.ietf.org/rfc/rfc2131.txt?number=2131
  "Dynamic Host Configuration Protocol"（"动态主机配置协议"）。

- http://www.ietf.org/rfc/rfc2132.txt?number=2132
  "DHCP Options and BOOTP Vendor Extensions"（"DHCP 选项与 BOOTP 厂商扩展"）。

- http://www.ietf.org/rfc/rfc2608.txt?number=2608
  "Service Location Protocol,Version2"（"服务定位协议，版本 2"）。

- http://www.ietf.org/rfc/rfc2219.txt?number=2219
  482 "Use of DNS Aliases for Network Services"（"为网络服务使用 DNS 别名"）。

注 3：本书中译本已由中国电力出版社出版。（编者注）

第21章

日志记录与使用情况跟踪

几乎所有的服务器和代理都会记录下它们所处理的 HTTP 事务摘要。这么做出于一系列的原因：跟踪使用情况、安全性、计费、错误检测，等等。本章简要介绍了日志记录，研究了通常会记录 HTTP 事务哪些方面的信息以及一些常见日志格式中所包含的内容。

21.1　记录内容

大多数情况下，日志的记录出于两种原因：查找服务器或代理中存在的问题（比如，哪些请求失败了），或者是生成 Web 站点访问方式的统计信息。统计数据对市场营销、计费和容量规划（比如，决定是否需要增加服务器或带宽）都非常有用。

可以把一个 HTTP 事务中所有的首部都记录下来，但对每天要处理数百万个事务的服务器和代理来说，这些数据的体积超大，很快就会失控。不应该记录实际上你并不感兴趣，甚至从来都不会去看一眼的数据。

通常，只记录事务的基本信息就行了。通常会记录下来的几个字段示例为：

- HTTP 方法；
- 客户端和服务器的 HTTP 版本；
- 所请求资源的 URL；
- 响应的 HTTP 状态码；
- 请求和响应报文的尺寸（包含所有的实体主体部分）；
- 事务开始时的时间戳；
483 - `Referer` 首部和 `User-Agent` 首部的值。

HTTP 方法和 URL 说明了请求试图做些什么——比如，GET 某个资源或 POST 某个定单。可以用 URL 来记录 Web 站点上页面的受欢迎程度。

版本字符串给出了与客户端和服务器有关的一些提示，在客户端和服务器之间出现一些比较奇怪或非预期的交互动作时，它会非常有用。比如，如果请求的失败率高于预期，那版本信息指向的可能是一个无法与服务器进行交互的新版浏览器。

HTTP 状态码说明了请求的执行状况：是否成功执行，认证请求是否失败，资源是否找到等（HTTP 状态码列表参见 3.2.2 节）。

请求 / 响应的大小和时间戳主要用于记账；就是记录流入、流出或流经应用程序的字节有多少。还可用时间戳将观察到的问题与当时发起的一些请求关联起来。

21.2 日志格式

目前已有一些日志格式的标准了。本节我们会讨论一些最常见的日志格式。大部分商用和开源的 HTTP 应用程序都支持以一种或多种常用格式进行日志记录。很多这样的应用程序都支持管理者配置日志格式，创建自定义的格式。

应用程序支持管理者使用这些更标准的格式的主要好处之一就在于，可以充分利用那些已构建好的工具处理这些日志，并产生基本的统计信息。有很多开源包和商用包都可用来压缩日志，以进行汇报。使用标准格式，应用程序及其管理员就都可以利用这些包了。

21.2.1 常见日志格式

现在，最常见的日志格式之一就是*常用日志格式*。这种日志格式最初由 NCSA 定义，很多服务器在默认情况下都会使用这种日志格式。可以将大部分商用及开源服务器配置为使用这种格式，有很多商用及免费工具都可辅助解析常用日志格式的文件。表 21-1 按序列出了常用日志格式中的字段。

表21-1　常用日志格式字段

字　段	描　述
remotehost	请求端机器的主机名或 IP 地址（如果没有配置服务器去执行反向 DNS 或无法查找请求端的主机名，就使用 IP 地址）
username	如果执行了 ident 查找，就是请求端已认证的用户名[a]
auth-username	如果进行了认证，就是请求端已认证的用户名
timestamp	请求的日期和时间
request-line	精确的 HTTP 请求行文本，GET /index.html HTTP/1.1
response-code	响应中返回的 HTTP 状态码
response-size	响应主体中的 `Content-Length`，如果响应中没有返回主体，就记录 0

a：RFC 931 描述了在此认证中使用的 ident 查找。ident 协议是在第 5 章介绍的。

484

例 21-1 列出了几个常见日志格式条目。

例 21-1　常见日志格式

```
209.1.32.44 - - [03/Oct/1999:14:16:00 -0400] "GET / HTTP/1.0" 200 1024
http-guide.com - dg [03/Oct/1999:14:16:32 -0400] "GET / HTTP/1.0" 200 477
http-guide.com - dg [03/Oct/1999:14:16:32 -0400] "GET /foo HTTP/1.0" 404 0
```

在这些例子中，字段的分配如下所示。

字段1	条目1	条目2	条目2
remotehost	209.1.32.44	http-guide.com	http-guide.com
username	< 空 >	< 空 >	< 空 >
auth-username	< 空 >	dg	dg
timestamp	03/Oct/1999:14:16:00 -0400	03/Oct/1999:14:16:32 -0400	03/Oct/1999:14:16:32 -0400
request-line	GET / HTTP/1.0	GET / HTTP/1.0	GET /foo HTTP/1.0
response-code	200	200	404
response-size	1024	477	0

注意，remotehost 字段可以是 http-guide.com 那样的主机名，也可以是 209.1.32.44 这样的 IP 地址。

第二个（username）和第三个（auth-username）字段之间的破折号说明字段为空。这说明要么是没有进行 ident 查找（第二个字段为空），要么是没有进行认证（第三个字段为空）。

21.2.2 组合日志格式

另一种常用日志格式为组合日志格式（Combined Log Format），例如 Apache 服务器就支持这种格式。组合日志格式与常用日志格式很类似。实际上，它就是常用日志格式的精确镜像，只是添加了（表 21-2 中列出的）两个字段。User-Agent 字段用于说明是哪个 HTTP 客户端应用程序在发起已被记录的请求，而 Referer 字段则提供
485 了更多与请求端在何处找到这个 URL 的有关信息。

表21-2 新加的组合日志格式字段

字 段	描 述
Referer	Referer 首部的内容
User-Agent	User-Agent 首部的内容

例 21-2 给出了一个组合日志格式的条目。

例 21-2 组合日志格式

```
209.1.32.44 - - [03/Oct/1999:14:16:00 -0400] "GET / HTTP/1.0" 200 1024
"http://www.joes-hardware.com/"  "5.0: Mozilla/4.0 (compatible; MSIE
5.0; Windows 98)"
```

在例 21-2 中，Referer 字段和 User-Agent 字段的值如下所示。

字 段	值
Referer	http://www.joes-hardware.com/
User-Agent	5.0: Mozilla/4.0（兼容的，MSIE 5.0，Windows 98）

例 21-2 的组合日志格式条目示例中的前七个字段和常用日志格式中的完全一样（参见例 21-1 中的第一个条目）。两个新字段 Referer 和 User-Agent 附加在日志条目的末尾。

21.2.3　网景扩展日志格式

网景进入商用 HTTP 应用程序领域时，为其服务器定义了很多其他 HTTP 应用程序开发者已接纳的日志格式。网景的格式是基于 NCSA 的常用日志格式的，但它们扩展了该格式，以支持与代理和 Web 缓存这样的 HTTP 应用程序相关的字段。

网景扩展日志格式的前 7 个字段与常用日志格式中的那些字段完全相同（参见表 21-1）。表 21-3 按序列出了网景扩展日志格式引入的新字段。

表21-3　网景扩展日志格式新加的字段

字　　段	描　　述
proxy-response-code	如果事务处理经过了某个代理，就是从服务器传往代理的 HTTP 响应码
proxy-response-size	如果事务处理经过了某个代理，就是发送给代理的服务器响应实体的 `Content-Length`
client-request-size	发给代理的客户端请求的所有主体或实体的 `Content-Length`
proxy-request-size	如果事务处理经过了某个代理，就是代理发往服务器的请求的所有主体或实体的 `Content-Length`
client-request-hdr-size	以字节为单位的客户端请求首部的长度
proxy-response-hdr-size	如果事务处理经过了某个代理，就是以字节为单位的、发送给请求端的代理响应首部的长度 486
proxy-request-hdr-size	如果事务处理经过了某个代理，就是以字节为单位的、发送给服务器的代理请求首部的长度
server-response-hdr-size	以字节为单位的，服务器响应首部的长度
proxy-timestamp	如果事务处理经过了某个代理，就是请求和响应经过代理传输所经过的时间（单位为秒）

例 21-3 给出了一个网景扩展日志格式的条目。

例 21-3　网景扩展日志格式

```
209.1.32.44 - - [03/Oct/1999:14:16:00-0400] "GET / HTTP/1.0" 200 1024
200 1024 0 0 215 260
279 254 3
```

在这个例子中，扩展字段的值如下所示。

字　段	值
proxy-response-code	200
proxy-response-size	1024
client-request-size	0
proxy-request-size	0
client-request-hdr-size	215
proxy-response-hdr-size	260
proxy-request-hdr-size	279
server-response-hdr-size	254
proxy-timestamp	3

例 21-3 中网景扩展日志格式条目示例中的前 7 个字段是常用日志格式条目示例的镜像（参见例 21-1 中的第一个条目）。

21.2.4　网景扩展2日志格式

另一种网景日志格式，网景扩展 2 日志格式采用了扩展日志格式，并添加了一些与 HTTP 代理和 Web 缓存应用程序有关的附加信息。这些附加字段有助于更好地描绘 HTTP 客户端和 HTTP 代理应用程序间的交互图景。

网景扩展 2 日志格式是基于网景扩展日志格式的，初始字段与表 21-3 中列出的字段
487 完全相同（它也是表 21-1 中常用日志格式字段的扩展）。

表 21-4 按序列出了网景扩展 2 日志格式新加的字段。

表21-4　附加的网景扩展2日志格式字段

字　段	描　述
route	代理用来向客户端发送请求的路径（参见表 21-5）
client-finish-status-code	客户端完成状态码。说明了发送给代理的客户端请求是成功完成（FIN）了，还是被打断了（INTR）
proxy-finish-status-code	代理完成状态码。说明代理发送给服务器的请求是成功完成（FIN）了，还是被打断了（INTR）
cache-result-code	缓存结果代码；说明缓存是如何响应请求的[a]

a：表 21-7 列出了网景的缓存结果代码。

例 21-4 给出了一个网景扩展 2 日志格式的条目。

例 21-4　网景扩展 2 日志格式

```
209.1.32.44 - - [03/Oct/1999:14:16:00-0400] "GET / HTTP/1.0" 200 1024
200 1024 0 0 215 260
```

```
279 254 3 DIRECT FIN FIN WRITTEN
```

这个例子中扩展字段的值如下所示。

字　段	值
route	DIRECT
client-finish-status-code	FIN
proxy-finish-status-code	FIN
cache-result-code	WRITTEN

例 21-4 的网景扩展 2 日志格式条目中的前 16 个字段就是网景扩展日志格式示例条目的镜像（参见例 21-3）。

表 21-5 列出了有效的网景路由代码。

表21-5　网景路由代码

值	描　述
DIRECT	资源是直接从服务器上获取的
PROXY(host:port)	资源是通过代理“host:port”获取的
SOCKS(socks:port)	资源是通过 SOCKS 服务器“host:port”获取的

表 21-6 列出了有效的网景完成代码。

表21-6　网景完成状态码

值	描　述	
-	请求未开始	
FIN	请求成功完成	
INTR	请求被客户端中断或被代理 / 服务器终止	488
TIMEOUT	请求被代理 / 服务器超时	

表 21-7 列出了有效的网景缓存代码。[1]

表21-7　网景缓存代码

代　码	描　述
-	资源不可缓存
WRITTEN	资源被写入了缓存
REFRESHED	资源被缓存，且被刷新了
NO-CHECK	返回已缓存的资源，未进行新鲜性检查

注 1：表 21-7 列出了网景缓存结果代码。

（续）

代　码	描　述
UP-TO-DATE	返回已缓存的资源，进行了新鲜性检查
HOST-NOT-AVAILABLE	返回已缓存的资源。由于远程服务器不可用，所以未进行新鲜性检查
CL-MISMATCH	未将资源写入缓存。由于 `Content-Length` 与资源尺寸不匹配，放弃了写操作
ERROR	因为出错，资源未被写入缓存。比如，出现了超时，或客户端放弃了此事务

与很多其他 HTTP 应用程序一样，网景应用程序也有其他的日志格式，包括一种灵活日志格式和一种管理者输出自定义日志字段的方式。这些格式给予管理者更大的控制权，并可以选择在日志中报告 HTTP 事务处理的哪些部分（首部、状态、尺寸等），以自定义其日志。

由于很难预测管理者希望从其日志中获取哪些信息，才添加了管理者配置自定义格式的能力。很多其他的代理和服务器都有发布自定义日志的能力。

21.2.5　Squid代理日志格式

Squid 代理缓存（http://www.squid-cache.org）是 Web 上一个很古老的部分。其起源可以回溯到一个早期的 Web 代理缓存项目（ftp://ftp.cs.colorado.edu/pub/techreports/schwartz/Harvest.Conf.ps.Z）。Squid 是开源社团多年来扩展增强的一个开源项目。有很多工具可以用来辅助管理 Squid 应用程序，包括一些有助于处理、审核及开发其日志的工具。很多后继代理缓存都为自己的日志使用了 Squid 格式，这样才能更
489 好地利用这些工具。

Squid 日志条目的格式相当简单。表 21-8 总结了该日志格式的字段。

表21-8　Squid日志格式字段

字　段	描　述
timestamp	请求到达时的时间戳，是从格林尼治标准时间 1970 年 1 月 1 日开始的秒数
time-elapsed	请求和响应通过代理传输所经历的时间（以毫秒为单位）
host-ip	客户端（请求端）主机的 IP 地址
result-code/status	result 字段是 Squid 类型的，用来说明在此请求过程中代理采取了什么动作；[a] code 字段是代理发送给客户端的 HTTP 响应代码
size	代理响应客户端的字节长度，包括 HTTP 响应首部和主体
method	客户端请求的 HTTP 方法

a：表 21-9 列出了各种结果代码及其含义。

（续）

字　段	描　述
url	客户端请求中的 URL[b]
rfc931-ident[c]	客户端经过认证的用户名[d]
hierarchy/from	与网景格式中的 route 字段一样，hierarchy 字段说明了代理向客户端发送请求时经由的路径。[e]from 字段说明了代理用来发起请求的服务器的名称
content-type	代理响应实体的 `Content-Type`

b：回顾第 2 章，代理通常会记录下整条请求 URL，这样如果 URL 中有用户和密码组件，代理就会在不经意间记录下此信息。

c：rfc931-ident、hierarchy/from 和 content-type 字段是在 Squid 1.1 中添加的。早期版本中都没有这些字段。

d：RFC 931 描述了这种认证方式中使用的 ident 查找。

e：http://squid.nlanr.net/Doc/FAQ/FAQ-6.html#ss6.6 列出了所有有效的 Squid hierarchy 代码。

例 21-5 给出了一个 Squid 日志格式条目的例子。

例 21-5　Squid 日志格式

```
99823414 3001 209.1.32.44 TCP_MISS/200 4087 GET http://www.joes-
hardware.com - DIRECT/
proxy.com text/html
```

这些字段的值如下所示。

字　段	值
timestamp	99823414
time-elapsed	3001
host-ip	209.1.32.44
action-code	TCP_MISS
status	200
size	4087
method	GET
URL	http://www.joes-hardware.com
RFC 931 ident	-
hierarchy	DIRECT[a]
from	proxy.com
content-type	text/html

490

a：DIRECT Squid hierarchy 值与网景日志格式中的 DIRECT route 值一样。

表 21-9 列出了各种 Squid 结果代码。[2]

注 2：这些行为代码中有几个通常是处理 Squid 代理缓存的内部行为，因此其他使用了 Squid 日志格式的代理并没有使用全部的代码。

表21-9　Squid结果代码

行　为	描　述
TCP_HIT	资源的有效副本是由缓存提供的
TCP_MISS	资源不在缓存中
TCP_REFRESH_HIT	资源在缓存中，但需要进行新鲜性检查。代理与服务器再次验证了资源，发现缓存中的副本确实还是新鲜的
TCP_REF_FAIL_HIT	资源在缓存中，但需要进行新鲜性检查。但再验证失败了（可能是代理无法连接到服务器），因此返回的是“过期”的资源
TCP_REFRESH_MISS	资源在缓存中，但需要进行新鲜性检查。在与服务器进行验证的时候，代理得知缓存中的资源过期了，并收到一个新的副本
TCP_CLIENT_REFRESH_MISS	请求端发送了一个 `Pragma: no-cache`，或类似的 `Cache-Control` 指令，命令代理必须去获取资源
TCP_IMS_HIT	请求端发布了一个条件请求，对资源的已缓存副本进行验证
TCP_SWAPFAIL_MISS	代理认为资源位于缓存中，但出于某些原因无法访问该资源
TCP_NEGATIVE_HIT	返回已缓存的响应，但响应是否定的已缓存响应。Squid 支持对资源错误信息的缓存（比如，缓存 404 Not Found 响应）。这样，如果有多条对某无效资源的请求都经过这个代理缓存，就可以由这个代理缓存提供错误信息
TCP_MEM_HIT	资源的有效副本是由缓存提供的，资源位于代理缓存的内存中（与必须访问磁盘才能获取已缓存资源的方式相反）
TCP_DENIED	对此资源的请求被否决了，可能是请求端没有请求此资源的权利
TCP_OFFLINE_HIT	所请求的资源是在离线状态下从缓存中解析出来的。Squid（或另一个使用此格式的代理）处于离线模式时，资源是未经验证的
UDP_*	UDP_* 代码说明请求是通过到代理的 UDP 接口收到的。HTTP 通常会使用 TCP 传输协议，因此这些请求使用的都不是 HTTP 协议[a]
UDP_HIT	资源的有效副本是由缓存提供的
UDP_MISS	资源不在缓存中
UDP_DENIED	对此资源的请求被否决了，可能是由于请求端没有请求此资源的权限
UDP_INVALID	代理收到的请求是无效的
UDP_MISS_NOFETCH	Squid 在特定的操作模式下，或在缓存常见错误的缓存中使用。会返回缓存未命中，而且也没有获得资源
NONE	有时与错误信息一起记录
TCP_CLIENT_REFRESH	参见 TCP_CLIENT_REFRESH_MISS
TCP_SWAPFAIL	参见 TCP_SWAPFAIL_MISS
UDP_RELOADING	参见 UDP_MISS_NOFETCH

491

a：Squid 有自己的用于发起这些请求的协议：ICP。这是缓存到缓存的请求所使用的协议。更多信息请参见 http://www.squid-cache.org。

21.3 命中率测量

原始服务器通常会出于计费的目的保留详细的日志记录。内容提供者需要知道URL 的受访频率，广告商需要知道广告的出现频率，网站作者需要知道所编写的内容的受欢迎程度。客户端直接访问 Web 服务器时，日志记录可以很好地跟踪这些信息。

但是，缓存服务器位于客户端和服务器之间，用于防止服务器同时处理大量访问请求[3]（这正是缓存的目的）。缓存要处理很多 HTTP 请求，并在不访问原始服务器的情况下满足它们的请求，服务器中没有客户端访问其内容的记录，导致日志文件中出现遗漏。

由于日志数据会遗失，所以，内容提供者会对其最重要的页面进行*缓存清除*（cache bust）。缓存清除是指内容提供者有意将某些内容设置为无法缓存，这样，所有对此内容的请求都会被导向原始服务器。[4] 于是，原始服务器就可以记录下访问情况了。不使用缓存可能会生成更好的日志，但会减缓原始服务器和网络的请求速度，并增加其负荷。

由于代理缓存（及一些客户端）都会保留自己的日志，所以如果服务器能够访问这些日志（或者至少有一种粗略的方式可以判断代理缓存会以怎样的频率提供其内容），就可以避免使用缓存清除。命中率测量协议是对 HTTP 的一种扩展，它为这个问题提供了一种解决方案。命中率测量协议要求缓存周期性地向原始服务器汇报缓存访问的统计数据。 492

RFC 2227 详细定义了命中率测量协议。本节将详细介绍此协议。

21.3.1 概述

命中率测量协议定义了一种 HTTP 扩展，它提供了一些基本的功能，缓存和服务器可以实现这些功能来共享访问信息，规范已缓存资源的可使用次数。

缓存给日志访问带来了问题，命中率测量并不是这个问题的完整解决方案，但它确实提供了一种基本方式，以获取服务器希望跟踪的度量值。命中率测量协议并没有（而且可能永远都不会）得到广泛的实现或应用。也就是说，在维护缓存性能增益的同时，像命中率测量这样的合作方案会给出一些提供精确访问统计信息的承诺。希望这会推动命中率测量协议的实现，而不是把内容标记为不可缓存的。

注 3：回想一下，几乎每个浏览器都会有一个缓存。

注 4：第 7 章说明了怎样将 HTTP 响应标记为不可缓存的。

21.3.2 Meter首部

命中率测量扩展建议使用新增加的首部 `Meter`，缓存和服务器可以通过它在相互间传输与用法和报告有关的指令，这与用来进行缓存指令交换的 `Cache-Control` 首部很类似。

表 21-10 列出了定义的各种指令和谁可以在 `Meter` 首部传输这些指令。

表21-10　命中率测量指令

指　　令	缩　　写	执　行　者	描　　述
will-report-and-limit	w	缓存	缓存可以报告使用情况并遵循服务器指定的所有使用限制
wont-report	x	缓存	缓存可以遵循使用限制，但不报告使用情况
wont-limit	y	缓存	缓存可以报告使用情况但不会限制使用
count	c	缓存	报告指令，以 `uses/reuses` 整数的形式说明。比如：count=2/4 [a]
max-uses	u	服务器	允许服务器指定某响应可被缓存使用的最大次数。比如：max-uses=100
max-reuses	r	服务器	允许服务器指定某响应可被缓存重用的最大次数。比如：max-reuses=100
do-report	d	服务器	服务器要求代理发送使用报告
dont-report	e	服务器	服务器不要求使用汇报
timeout	t	服务器	允许服务器指定对某资源进行计量的超时时间。缓存应该在指定的超时时间，或在此时间之前发送报告，允许有 1 分钟的误差。超时是以分钟为单位的。比如：timeout=60
493 wont-ask	n	服务器	服务器不需要任何计量信息

a：命中率测量定义了一个 use，用一个响应来满足请求，还定义了一个 reuse，对客户端请求进行再验证。

图 21-1 给出了一个执行中的命中率测量实例。事务的第一部分就是客户端和代理缓存之间一个普通的 HTTP 事务，但在代理请求中，要注意有插入的 `Meter` 首部和来自服务器的响应。这里，代理正在通知服务器它可以进行命中率测量，作为回应，服务器则请求代理报告它的命中次数。

从客户端的角度来看，请求正常结束了，代理开始代表服务器跟踪该请求资源的命中次数。稍后，代理会尝试与服务器再次验证资源。代理会在发送给服务器的条件请求中嵌入它跟踪记录的计量信息。

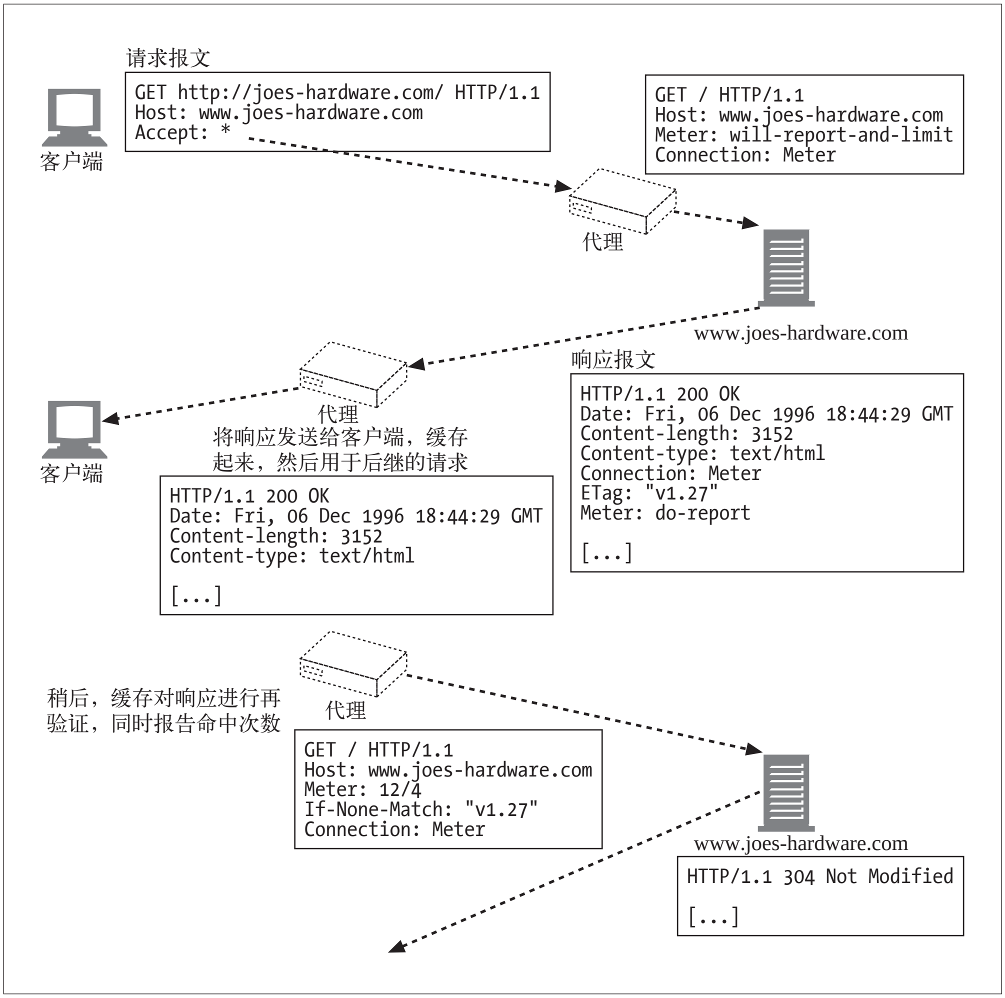


图 21-1　命中率测量示例

494

21.4　关于隐私的考虑

日志记录实际上就是服务器和代理执行的一项管理功能，所以整个操作对用户来说都是透明的。通常，用户甚至都不清楚他们的 HTTP 事务已被记录——实际上，很多用户可能甚至都不知道他们访问 Web 上的内容时是在使用 HTTP 协议。

Web 应用程序的开发者和管理者要清楚跟踪用户的 HTTP 事务可能带来的影响。他可以根据获取的信息收集很多有关用户的情况。很显然，这些信息可以用于不良目的——歧视、骚扰、勒索等。进行日志记录的 Web 服务器和代理一定要注意保护其终端用户的隐私。

有些情况下，比如在工作环境中，跟踪某用户的使用情况以确保他没有偷懒是可行的，但管理员也应该将监视大家事务处理的事情公之于众。

简单来说，日志记录对管理者和开发者来说都是很有用的工具。只是要清楚在没有获得用户许可，或在其不知情的情况下，使用记录其行为的日志可能会存在侵犯隐私的问题。

21.5 更多信息

更多有关日志记录的信息，请参见以下资源。

- http://httpd.apache.org/docs/logs.html
  “Apache HTTP Server: Log Files”（“Apache HTTP 服务器：日志文件”）。Apache HTTP 服务器项目网站。

- http://www.squid-cache.org/Doc/FAQ/FAQ-6.html
  “Squid Log Files”（“Squid 日志文件”）。Squid 代理缓存网站。

- http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format
  “Logging Control in W3C httpd”（“W3C httpd 中的日志记录控制”）。

- http://www.w3.org/TR/WD-logfile.html
  “Extended Log File Format”（“扩展日志文件格式”）。

- http://www.ietf.org/rfc/rfc2227.txt
  RFC 2227，J. Mogul 和 P. Leach 编写的“Simple Hit-Metering and Usage-
  495 Limiting for HTTP”（“简单的 HTTP 命中率测量和使用限制”）。

第六部分

附　录

本书附录集中包含了一些有用的参考表格、背景信息以及关于 HTTP 结构和实现各种主题的指南。

- 附录 A　URI 方案
- 附录 B　HTTP 状态码
- 附录 C　HTTP 首部参考
- 附录 D　MIME 类型
- 附录 E　Base-64 编码
- 附录 F　摘要认证
- 附录 G　语言标记
- 附录 H　MIME 字符集注册表

496
~
498

附录A

URI方案

已定义的 URI 方案有很多，但常用的并不多。一般来说，有相关的 RFC 对其解释说明的 URI 方案更常用一些，但确实也有少数由主导软件公司（特别是 Netscape 和 Microsoft）开发，但未被正式发布的方案得到了广泛应用。

W3C 维护了一个 URI 方案列表，可以通过以下地址访问：

http://www.w3.org/Addressing/schemes.html

IANA 也维护了一个 URL 方案的列表，网址是：

http://www.iana.org/assignments/uri-schemes

表 A-1 介绍了部分已经提出的和正在使用的方案。注意，表中大约有 90 个方案，其中很多都没有得到广泛应用，而且有些已被废弃。

表A-1　在W3C注册的URI方案

方　　案	描　　述	RFC
about	研究浏览器各方面特性的 Netscape 方案。比如说，使用 about 自身的效果就跟选择 Navigator 的 Help 菜单中的 About Communicator 一样，about:cache 显示的就是磁盘缓存的统计数据，about:plugins 显示的是与已配置的插件有关的信息。其他浏览器，比如微软的 Internet Explorer，也使用了这个方案	
acap	应用程序配置访问协议	2244
afp	用于使用 AFP（Apple Filing Protocol，苹果文件协议）提供的文件共享服务，是作为已过期的 IETF draft-ietf-svrloc-afp-service-01.txt 的一部分定义的	
afs	保留，以备 Andrew 文件系统将来使用	
callto	初始化微软的 NetMeeting 会议的会话，比如：callto:ws3.joes-hardware.com/joe@joes-hardware.com	
chttp	Real 网络公司定义的 CHTTP 缓存协议。RealPlayer 没有缓存 HTTP 传输的所有条目。作为一种替代方式，可以在文件的 URL 中使用 chttp:// 代替 http:// 来说明要缓存的文件。RealPlayer 在 SMIL 文件中读到 CHTTP URL 时，首先会查看磁盘的缓存中是否有该文件。如果没有此文件，就通过 HTTP 请求此文件，并将其存储在自己的缓存中	
cid	在电子邮件中通过 [MIME] 传送 Web 页面及其相关图片时，需要一个 URL 方案允许 HTML 引用报文中所含图片或其他数据 Content-ID URL，即 cid，就用于这个目的	2392 2111
clsid	允许引用微软的 OLE/COM 类。用于向 Web 页面中插入活动对象	

499

（续）

方　　案	描　　述	RFC
data	允许将一些小的常量数据条目作为“即时”数据包含在内。这个 URL 会将 text/plain 字符串 A brief note 编码为：data:A%20brief%20note	2397
date	支持日期的方案建议，比如 date:1999-03-04T20:42:08	
dav	为了确保基于此规范的互操作的正确性，IANA 必须保留以 DAV: 和 opaquelocktoken: 开头的 URI 名字空间，供这个规范和它的修订版本以及相关的 WebDAV 规范使用	2518
dns	供 REBOL 软件使用。 参见 http://www.rebol.com/users/valurl.html	
eid	外部 ID 方案提供了一种机制，本地应用程序可以通过这种机制引用通过其他非 URL 方案获取的数据。这个方案试图提供一种通用的转义机制，以便那些无法提出自己方案的专业应用程序访问信息。这个 URI 的使用方式存在争议。参见 http://www.ics.uci.edu/pub/ietf/uri/draft-finseth-url-00.txt	
fax	方案 fax 描述了一条连接，此连接连至可处理电传的终端（传真机）	2806
file	在特定主机上标识出可访问的文件。其中可以包含主机名，但这个方案的特殊性在于它没有为此类文件指定因特网协议或访问方式；这样，它在主机间网络协议中的效用就会受到限制	1738
finger	finger URL 的格式如下：*finger*://*host*[:*port*][/<*request*>]。 <request> 必须与 RFC 1288 的请求格式一致。参见 http://www.ics.uci.edu/pub/ietf/uri/draft-ietf-uri-url-finger-03.txt	
freenet	获取中迅互联分布式信息系统中信息所用的 URI。参见 http://freenet.sourceforge.net	
ftp	文件传输协议方案	1738
gopher	古老的 gopher 协议	1738
gsm-sms	用于 GSM 移动电话短信业务的 URI。	
h323，h324	多媒体会议的 URI 方案。 参见 http://www.ics.uci.edu/pub/ietf/uri/draft-cordell-sg16-conv-url-00.txt	
hdl	Handle 是个被广泛应用的系统，用于分配、管理数字对象和因特网上的其他资源，并将其解析成名为 handles 的永久标识符。可以将 handle 作为 URN 使用。参见 http://www.handle.net	
hnews	HNEWS 是 NNTP 新闻协议的一个 HTTP 隧道变体。hnews URL 语法设计与当前常用的新闻 URL 方案兼容。参见 http://www.ics.uci.edu/pub/ietf/uri/draft-stockwell-hnews-url-00.txt	
http	HTTP 协议。更多信息请参见本书	2616

500

（续）

方　案	描　述	RFC
https	SSL 上的 HTTP。 参见 http://sitesearch.netscape.com/eng/ssl3/draft302.txt	
iiopIoc	CORBA 扩展。可互操作的名字服务定义了一种 URL 格式的对象引用——iiopIoc，可以将其输入一个程序中，以获取包括名字服务在内的已定义远程服务。比如，以下 iiopIoc 标识符：iiopIoc://www.omg.org/NameService 就表示运行在 IP 地址与域名 www.omg.org 相对应的机器上的 CORBA 名字服务。参见 http://www.omg.org	
ilu	ILU（Inter-Language Unification，跨语言统一）系统是一个多语言对象接口系统。ILU 提供的对象接口隐藏了不同语言、不同地址空间和不同操作系统之间的实现差异。可以通过 ILU，用经过良好说明的语言无关接口来构建多语言的面向对象库（类库）。还可以将其用于实现分布式系统。 参见 ftp://parcftp.parc.xerox.com/pub/ilu/ilu.html	
imap	IMAP URL 方案用于分配 IMAP 服务器、邮箱、报文、MIME 主体 [MIME]，在因特网主机上搜索可以通过 IMAP 协议访问的程序	2192
IOR	CORBA 的互操作对象引用。 请参见 http://www.omg.org	
irc	irc URL 方案用于表示 IRC（Internet Relay Chat，因特网中继聊天）服务器或者 IRC 服务器上的独立实体（信道或人）。 参见 http://www.w3.org/Addressing/draft-mirashi-url-irc-01.txt	
isbn	建议用于 ISBN 书籍参考的方案。 参见 http://lists.w3.org/Archives/Public/www-talk/1991NovDec/0008.html	
java	用来识别 Java 类	
javascript	网景的浏览器会处理 javascript URL，如果冒号（：）后面有表达式，则计算它的值，只要表达式字符串的值不是未定义的，就会加载包含了这个值的页面	
jdbc	用于 Java SQL API	
ldap	允许因特网客户端直接访问 LDAP 协议	2255
lid	本地标识符方案。 参见 draft-blackketter-lid-00	
lifn	UTK 开发的批量文件分发分布式存储系统所使用的 lifn（location-independent file name，位置无关文件名）	
livescript	JavaScript 的曾用名	

（续）

方　　案	描　　述	RFC
lrq	参见 h323	
mailto	URL 方案 mailto 用于访问单个用户或服务的因特网邮件地址	2368
mailserver	1994 ～ 1995 年的老建议，支持将整条报文都编码到一个 URL 中去，这样（比如说）URL 可以自动向邮件服务器发送订阅邮件列表的电子邮件了 501	
md5	MD5 是一种密码校验和	
mid	mid 方案用电子邮件报文的 message-id（一部分）来引用一个特定的报文	2392 2111
mocha	参见 javascript	
modem	modem 方案描述了一条连接，连接到能够处理输入数据呼叫的终端上去	2806
mms、mmst、mmsu	MMS（Microsoft Media Server，微软媒体服务器）以流方式传送 ASF（Active Streaming Format，活动流格式）文件时使用的方案。强制使用 UDP 传输时，使用 mmsu 方案。强制使用 TCP 传输时，使用 mmst 方案	
news	news URL 方案指的是 USENET 新闻中的新闻组，或独立的文章。news URL 使用下列两种格式之一： *news:<newsgroup-name>* 或 *news:<message-id>*	1738 1036
nfs	指的是 NFS 服务器上的文件和目录	2224
nntp	另一种引用 news 文章的方法，指定 NNTP 服务器上的 news 文章时很有用。nntp URL 看起来如下所示： nntp://*<host>*:*<port>*/*<newsgroup-name>*/*<article-num>* 注意，尽管 nntp URL 为文章资源指定了唯一的位置信息，但现在因特网上大部分 NNTP 服务器都配置为只允许从本地客户端访问，nntp URL 因此就无法指定全球可访问的资源了。因此，URL 的 news 格式更常作为识别新闻性文章的一种方式	1738 977
opaquelocktoken	以 URI 形式表示的 WebDAV 锁定令牌，用于标识特定锁的。每个成功的 LOCK 操作都会在响应主体的 lockdiscovery 特性中返回锁定令牌，也可以通过资源的锁发现操作找到它。参见 RFC 2518	
path	path 方案定义了一个统一的层次化命名空间。在这个空间中，path URN 就是由一些组件和可选的不透明字符串组成的序列。 参见 http://www.hypernews.org/~liberte/www/path.html	
phone	在“电话的 URL”中使用。在 RFC 2806 中被 tel: 取代	
pop	POP URL 指定了一个 POP 电子邮件服务器，一个可选的端口号、认证机制、认证 ID 和 / 或授权 ID	2384

（续）

方　　案	描　　述	RFC
pnm	Real 网络公司的流媒体协议	
pop3	POP3 URL 方案允许 URL 指定一个 POP3 服务器，允许其他协议使用通用的“用于邮件访问的 URL”取代对 POP3 的显式引用。在已过期的 draft-earhart-url-pop3-00.txt 中定义	
printer	用于服务定位标准的抽象 URL。 参见 draft-ietf-srvloc-printer-scheme-02.txt	
prospero	通过 prospero 目录服务访问的名字资源	1738
res	微软的方案，指定了一个要从某模块中获取的资源。包含一个字符串或数字资源类型和一个字符串或数字 ID	
rtsp	实时流协议，是 Real 网络公司现代流媒体控制协议的基础协议	2326
502 rvp	RVP 集合点协议的 URL，用于在某计算机网络上发布用户到来的通知。参见 draft-calsyn-rvp-01	
rwhois	RWhois 是在 RFC 1714 和 RFC 2167 中定义的因特网目录访问协议。RWhois URL 将 rwhois 的直接访问权赋予了客户端。 参见 http://www.rwhois.net/rwhois/docs/	
rx	一种结构，允许远程图像应用程序在 Web 页面中显示数据。 参见 http://www.w3.org/People/danield/papers/mobgui/	
sdp	sdp（session description protocol，会话描述协议）URL。参见 RFC 2327	
service	service 方案可以为任意网络服务提供访问信息。这些 URL 为基于客户端的网络软件提供了一种可扩展的框架，以获取使用网络服务所需的配置信息	2609
sip	sip* 族方案用于建立使用 sip（session initiation protocol，会话发起协议）的多媒体会议	2543
shttp	S-HTTP 是 HTTP 的超集，用于保护 HTTP 连接，它提供了大量机制用以实现保密性、认证功能和完整性。S-HTTP 没有被广泛采用，主要是被 HTTPS（经过 SSL 加密的 HTTP）取代了。 参见 http://www.homeport.org/~adam/shttp.html	
snews	经 SSL 加密的 news	
STANF	用于可靠网络文件名的老建议。与 URN 有关。 参见 http://Web3.w3.org/Addressing/#STANF	
t120	参见 h323	
tel	通过电话网打电话的 URL	2806
telephone	用于 tel 的早期草案中	

（续）

方　　案	描　　述	RFC
telnet	指定了可能会被 Telnet 协议访问的交互式业务。Telent URL 的格式如下所示： *telnet://<user>:<password>@<host>:<port>/*	1738
tip	支持 TIP 原子化的因特网事务处理	2371 2372
tn3270	根据 ftp://ftp.isi.edu/in-notes/iana/assignments/url-schemes 保留	
tv	TV URL 命名了一个特定的电视广播信道	2838
uuid	UUID（通用唯一标识符）不包含与位置有关的信息。也称为 GUID（全球唯一标识符）。由一个 128 位的唯一 ID 组成。它和 URN 一样，不会随时间发生变化。要使用无法或不应该依赖于特定的物理根名字空间（比如一个 DNS 名称）的通用标识符时，UUID URI 是很有用的。 参见 draft-kindel-uuid-uri-00.txt	
urn	持久的、与位置无关的 URN	2141
vemmi	允许 VEMMI（万用多媒体接口）客户端软件和 VEMMI 终端连接 VEMMI 兼容的服务。VEMMI 是一种在线多媒体应用服务的国际标准	2122
videotex	允许 videotex 客户端软件或终端连接与 ITU-T 和 ETSI videotex 标准兼容的 videotex 服务。 参见 http://www.ics.uci.edu/pub/ietf/uri/draft-mavrakis-videotex-url-spec-01.txt	503
view-source	网景的 Navigator 的源码查看器。这些 view-source URL 可以显示用 JavaScript 生成的 HTML	
wais	广域信息服务——一种早期搜索引擎形式	1738
whois++	WHOIS++ 简单因特网目录协议的 URL。 参见 http://martinh.net/wip/whois-url.txt	1835
whodp	WhoDP（Widely Hosted Object Data Protocol，广泛托管对象数据协议）用于沟通大量动态、可重定位对象的当前位置和状态传递。WhoDP 程序通过“订阅”定位对象，接收与某对象有关的信息，并“发布”这些信息，控制此对象的位置和可见状态	
z39.50r，z39.50s	Z39.50 会话与检索 URL。Z39.50 是一种信息检索协议，不能很好地适用于主要为获取无状态数据而设计的检索模式。它将通用的用户查询设计为面向会话的多步任务，服务器在继续处理任务之前，会向客户端请求额外的参数，因此每一步任务都可能被临时挂起	2056 504

附录B

HTTP状态码

附录 B 是 HTTP 状态码及其含义的快速参考。

B.1　状态码分类

HTTP 状态码分为 5 类，如表 B-1 所示。

表B-1　状态码分类

总体范围	已定义范围	类　　别
100~199	100~101	信息
200~299	200~206	成功
300~399	300~305	重定向
400~499	400~415	客户端错误
500~599	500~505	服务器错误

B.2　状态码

表 B-2 是 HTTP/1.1 规范定义的所有状态码的快速参考，表中概述了每种状态码及其含义。3.4 节曾详细地介绍了这些状态码及其用法。

505

表B-2　状态码

状态码	原因短语	含　　义
100	Continue（继续）	收到了请求的起始部分，客户端应该继续请求
101	Switching Protocols（切换协议）	服务器正根据客户端的指示将协议切换成 `Update` 首部列出的协议
200	OK	服务器已成功处理请求
201	Created（已创建）	对那些要服务器创建对象的请求来说，资源已创建完毕
202	Accepted（已接受）	请求已接受，但服务器尚未处理
203	Non-Authoritative Information（非权威信息）	服务器已将事务成功处理，只是实体首部包含的信息不是来自原始服务器，而是来自资源的副本
204	No Content（没有内容）	响应报文包含一些首部和一个状态行，但不包含实体的主体内容
205	Reset Content（重置内容）	另一个主要用于浏览器的代码。意思是浏览器应该重置当前页面上所有的 HTML 表单
206	Partial Content（部分内容）	部分请求成功

（续）

状态码	原因短语	含　　义
300	Multiple Choices（多项选择）	客户端请求了实际指向多个资源的 URL。这个代码是和一个选项列表一起返回的，然后用户就可以选择他希望使用的选项了
301	Moved Permanently（永久搬离）	请求的 URL 已移走。响应中应该包含一个 Location URL，说明资源现在所处的位置
302	Found（已找到）	与状态码 301 类似，但这里的搬离是临时的。客户端应该用 `Location` 首部给出的 URL 对资源进行临时定位
303	See Other（参见其他）	告诉客户端应该用另一个 URL 获取资源。这个新的 URL 位于响应报文的 `Location` 首部
304	Not Modified（未修改）	客户端可以通过它们所包含的请求首部发起条件请求。这个代码说明资源未发生过变化
305	Use Proxy（使用代理）	必须通过代理访问资源，代理的位置是在 `Location` 首部中给出的
306	（未用）	这个状态码当前并未使用
307	Temporary Redirect（临时重定向）	和状态码 301 类似。但客户端应该用 `Location` 首部给出的 URL 对资源进行临时定位
400	Bad request（坏请求）	告诉客户端它发送了一条异常请求
401	Unauthorized（未授权）	与适当的首部一起返回，在客户端获得资源访问权之前，请它进行身份认证
402	Payment Required（要求付款）	当前此状态码并未使用，是为未来使用预留的
403	Forbidden（禁止）	服务器拒绝了请求
404	Not Found（未找到）	服务器无法找到所请求的 URL
405	Method Not Allowed（不允许使用的方法）	请求中有一个所请求的 URI 不支持的方法。响应中应该包含一个 `Allow` 首部，以告知客户端所请求的资源支持使用哪些方法
406	Not Acceptable（无法接受）	客户端可以指定一些参数来说明希望接受哪些类型的实体。服务器没有资源与客户端可接受的 URL 相匹配时可使用此代码
407	Proxy Authentication Required（要求进行代理认证）	和状态码 401 类似，但用于需要进行资源认证的代理服务器
408	Request Timeout（请求超时）	如果客户端完成其请求时花费的时间太长，服务器可以回送这个状态码并关闭连接
409	Conflict（冲突）	发出的请求在资源上造成了一些冲突

506

（续）

状态码	原因短语	含　　义
410	Gone（消失了）	除了服务器曾持有这些资源之外，与状态码 404 类似
411	Length Required（要求长度指示）	服务器要求在请求报文中包含 `Content-Length` 首部时会使用这个代码。发起的请求中若没有 `Content-Length` 首部，服务器是不会接受此资源请求的
412	Precondition Failed（先决条件失败）	如果客户端发起了一个条件请求，如果服务器无法满足其中的某个条件，就返回这个响应码
413	Request Entity Too Large（请求实体太大）	客户端发送的实体主体部分比服务器能够或者希望处理的要大
414	Request URI Too Long（请求 URI 太长）	客户端发送的请求所携带的请求 URL 超过了服务器能够或者希望处理的长度
415	Unsupported Media Type（不支持的媒体类型）	服务器无法理解或不支持客户端所发送的实体的内容类型
416	Requested Range Not Satisfiable（所请求的范围未得到满足）	请求报文请求的是某范围内的指定资源，但那个范围无效，或者未得到满足
417	Expectation Failed（无法满足期望）	请求的 `Expect` 首部包含了一个预期内容，但服务器无法满足
500	Internal Server Error（内部服务器错误）	服务器遇到了一个错误，使其无法为请求提供服务
501	Not Implemented（未实现）	服务器无法满足客户端请求的某个功能
502	Bad Gateway（网关故障）	作为代理或网关使用的服务器遇到了来自响应链中上游的无效响应
503	Service Unavailable（未提供此服务）	服务器目前无法为请求提供服务，但过一段时间就可以恢复服务
504	Gateway Timeout（网关超时）	与状态码 408 类似，但是响应来自网关或代理，此网关或代理在等待另一台服务器的响应时出现了超时
505	HTTP Version Not Supported（不支持的 HTTP 版本）	服务器收到的请求是以它不支持或不愿支持的协议版本表示的

507

附录C

HTTP首部参考

回想起第一个 HTTP 版本——版本 0.9，还是挺有趣的，因为它没有定义任何首部。尽管这样肯定存在弊端，但不得不为其简洁的优雅而啧啧称奇。

好吧，回到现实中来。现在有很多的 HTTP 首部，有一些是规范中定义的，还有一些是对规范的扩展。本附录提供了一些有关这些正式首部和扩展首部的背景知识，你还可以将其作为本书各种首部的索引使用，说明了这些首部的概念和特性是在正文的什么地方讨论的。这些首部大部分都很简单、直接，是它们之间或者与 HTTP 其他特性之间的交互使得事情变得比较复杂。本附录为所列首部提供了一些背景知识，并指导用户参阅书中详细讨论的对应章节。

本附录列出的首部是从 HTTP 规范、相关文档和我们自己使用 HTTP 报文和因特网上各种服务器和客户端的经验中提取出来的。

这个列表远远称不上完备。Web 中还有很多其他的扩展首部，更别说私有内部网络中使用的那些首部了。尽管如此，我们已经使这个表尽可能地完整了。当前的 HTTP/1.1 规范和官方首部及其规范描述参见 RFC 2616。

Accept

客户端用 Accept 首部来通知服务器可以接受哪些媒体类型。Accept 首部字段的值是客户端可以使用的媒体类型列表。如果 Web 浏览器无法显示 Web 上所有的多媒体对象类型，就可以在请求中包含 Accept 首部，这样浏览器就不会去下载它无法使用的视频或其他对象类型了。

为了防止服务器有多种版本的媒体类型，还可以在 Accept 首部字段中包含一个质量值（q 值）列表，用以告知服务器它优选哪种媒体类型。有关内容协商和 q 值的
508 完整讨论参见第 17 章。

类型 请求首部

注释 “*”是个特殊值用来通配媒体类型。比如，“*/*”表示所有类型，“image/*”表示所有的图片类型。

举例

```
Accept: text/*, image/*
Accept: text/*, image/gif, image/jpeg; q=1
```

Accept-Charset

客户端用 Accept-Charset 首部来通知服务器它可以接受哪些字符集或哪些是优选字符集。这个请求首部的值是个字符集列表和所列字符集可能的质量值。当服务器上有以多种可接受字符集表示的文档时，可以通过质量值告知服务器哪个字符集是优选的。有关内容协商和 q 值的完整讨论参见第 17 章。

类型 请求首部

注释 与 Accept 首部一样，“*”是个特殊字符。如果有“*”，就表示除了显式地用值设置的字符集之外的所有字符集。如果没有“*”，那么值字段中没有设置的所有字符集的 q 值都默认为零，这不包括字符集 iso-latin-1，它的默认值为 1。

基本语法 Accept-Charset: 1# ((charset | "*") [";" "q" "=" qvalue])

举例 Accept-Charset: iso-latin-1

Accept-Encoding

客户端用 Accept-Encoding 首部来告知服务器它可以接受哪些编码方式。如果服务器所持有的内容是经过编码的（可能是压缩过的），这个请求首部可以告诉服务器客户端是否会接受它。第 17 章探讨了 Accept-Encoding 首部。

类型 请求首部

基本语法 Accept-Encoding: 1# ((content-coding | "*") [";" "q" "=" qvalue])

举例 Accept-Encoding:[1]

Accept-Encoding:gzip

Accept-Encoding: compress;q=0.5,

gzip;q=1 509

Accept-Language

和其他 Accept 首部一样，客户端可以通过 Accept-Language 请求首部通知服务器可接受或优选哪些语言（比如，内容所使用的自然语言）。第 17 章详尽介绍了 Accept-Language 首部。

类型 请求首部

基本语法 Accept-Language: 1# (language-range [";" "q" "=" qvalue])

language-range = ((1*8ALPHA * ("-" 1*8ALPHA)) | "*")

举例 Accept-Language: en

Accept-Language: en;q=0.7, en-gb;q=0.5

注 1：这并不是印刷错误。它指的是身份编码——也就是未编码的内容。如果提供了空的 Accept-Encoding 首部，就说明只能接受未编码的内容。

Accept-Ranges

Accept-Ranges 首部与其他 Accept 首部不同——它是服务器使用的一种响应首部，用来告知客户端它们是否接受请求资源的某个范围。如果这个首部有赋值的话，这个值就说明了服务器允许对指定资源的哪个范围类型进行访问。

客户端可以在没有收到这个首部的情况下，对某资源发起范围请求。如果服务器不支持对那个资源的范围请求，可以以适当的状态码进行响应[2]，将 Accept-Ranges 的值设置为 none。服务器可以为普通请求发送 none 值，这样客户端以后就不会发起范围请求了。

第 17 章完整介绍了 Accept-Ranges 首部。

类型　响应首部

基本语法　`Accept-Ranges: 1# range-unit | none`

举例　`Accept-Ranges: none`

`Accept-Ranges: bytes`

Age

Age 首部可以告诉接收端响应已产生了多长时间。对于原始服务器是在多久之前产生的响应或是在多久之前向原始服务器再次验证响应而言，这是发送端所做的最好的猜测。首部的值是发送端所做的猜测，以秒为单位递增。更多有关 Age 首部的内容参见第 7 章。

类型　响应首部

510 **注释**　HTTP/1.1 缓存必须在发送的每条响应中都包含一个 Age 首部。

基本语法　`Age: delta-seconds`

举例　`Age: 60`

Allow

Allow 首部用于通知客户端可以对特定资源使用哪些 HTTP 方法。

类型　响应首部

注释　发送 405 Method Not Allowed 响应的 HTTP/1.1 服务器必须包含 Allow 首部。[3]

基本语法　`Allow: #Method`

注 2：比如，状态码 416（参见 3.4.4 节）。

注 3：更多有关状态码 405 的内容参见 3.4 节。

举例　　`Allow: GET, HEAD`

`Authorization`

`Authorization` 首部是由客户端发送的，用来向服务器回应自己的身体验证信息。客户端收到来自服务器的 401 Authentication Required 响应后，要在其请求中包含这个首部。这个首部的值取决于所使用的认证方案。有关 `Authorization` 首部的详细讨论参见第 14 章。

类型　　请求首部

基本语法　　`Authorization: authentication-scheme #authentication-param`

举例　　`Authorization: Basic YnJpYW4tdG90dHk6T3ch`

`Cache-Control`

`Cache-Control` 首部用于传输对象的缓存信息。这个首部是 HTTP/1.1 引入的比较复杂的首部之一。它的值是一个缓存指令，给出了与某个对象可缓存性有关的缓存特有指令。

第 7 章简要介绍了缓存，还说明了与这个首部有关的特定细节。

类型　　通用首部

举例　　`Cache-Control: no-cache` 511

`Client-ip`

`Client-ip` 首部是一些比较老的客户端和代理使用的扩展首部，用来传输运行客户端程序的计算机 IP 地址。

类型　　扩展请求首部

注释　　实现者应该了解这个首部的值所提供的信息是不安全的。

基本语法　　`Client-ip: ip-address`

举例　　`Client-ip: 209.1.33.49`

`Connection`

`Connection` 首部是个多少有点儿过载了的首部，它可能会把你搞晕。这个首部用于扩展了 keep-alive 连接的 HTTP/1.0 客户端，keep-alive 连接用于控制信息。[4] 在

注 4：更多有关 keep-alive 和持久连接的内容参见第 4 章。

HTTP/1.1 中，能识别出大部分较老的语义，但这个首部被赋予了新的功能。

在 HTTP/1.1 中，`Connection` 首部的值是一个标记列表，这些标记对应各种首部名称。应用程序收到带有 `Connection` 首部的 HTTP/1.1 报文后，应该对列表进行解析，并删除报文中所有在 `Connection` 首部列表中出现过的首部。它主要用于有代理网络环境，这样服务器或其他代理就可以指定不应传递的逐跳首部了。

`close` 是一个典型的标记值。这个标记意味着响应结束之后，连接会被关闭。不支持持久连接的 HTTP/1.1 应用程序要在所有请求和响应中插入带有 `close` 标记的 `Connection` 首部。

类型 通用首部

注释 虽然 RFC 2616 没有专门声明将 keep-alive 作为连接标记使用，有些（包括那些将 HTTP/1.1 作为版本号发送的）浏览器还是会在发起请求时使用它。

基本语法 `Connection: 1# (connection-token)`

512 **举例** `Connection: close`

Content-Base

服务器可以通过 `Content-Base` 首部为响应主体部分中要解析的 URL 指定一个基础 URL。[5]`Content-Base` 首部的值是一个绝对 URL，可以用来解析在实体内找到的相对 URL。

类型 实体首部

注释 RFC 2616 中没有定义这个首部。它是早期在 RFC 2068 中定义的，RFC 2068 是一个较早的 HTTP/1.1 规范草案，已经从官方规范中删除了。

基本语法 `Content-Base: absoluteURL`

举例 `Content-Base: http://www.joes-hardware.com/`

Content-Encoding

`Content-Encoding` 首部用于说明是否对某对象进行过编码。通过对内容进行编码，服务器可以在发送响应之前将其进行压缩。`Content-Encoding` 首部的值可以告诉客户端，服务器对对象执行过哪种或哪些类型的编码。有了这个信息，客户端就可以对报文进行解码了。

注 5：更多有关基础 URL 的信息参见 2.3 节。

有时服务器会对某个实体进行多种编码，在这种情况下，必须按照执行的顺序将编码列出来。

类型 实体首部

基本语法 `Content-Encoding: 1# content-coding`

举例 `Content-Encoding: gzip`

`Content-Encoding: compress, gzip`

Content-Language

`Content-Language` 首部用来告诉想要理解对象的客户端，应该理解哪种自然语言。比如说，一篇用法语编写的文档就应该有一个表示法语的 `Content-Language` 值。如果在响应中没有提供这个值，对象就是提供给所有用户的。首部值中有多种语言就说明对象适用于使用所列各种语言的用户。

这里需要说明的是，这个首部的值可能只表示了此对象目标用户的自然语言，而不是对象中包含的所有或者任意一种语言。而且，此首部并不局限于文本或书面数据 513
对象；图像、视频和其他媒体类型也可以用其目标用户的自然语言来标识。

类型 实体首部

基本语法 `Content-Language: 1# language-tag`

举例 `Content-Language: en`

`Content-Language: en, fr`

Content-Length

`Content-Length` 首部说明实体主体部分的长度或尺寸。如果对 HEAD HTTP 请求的响应报文中有这个首部，此首部的值就表示如果发送的的话，实体主体部分的长度（实际上并不发送主体）。

类型 实体首部

基本语法 `Content-Length: 1*DIGIT`

举例 `Content-Length: 2417`

Content-Location

`Content-Location` 首部包含在一个 HTTP 报文中，给出了与报文的实体部分相对应的 URL。对可能有多个 URL 的对象来说，响应报文中可以包含一个 `Content-Location` 首部，说明用来产生响应的对象的 URL。`Content-Location` 可以与所请求的 URL 不同。服务器通常会用它将客户端导向或重定向到一个新 URL 上去。

如果 URL 是相对的，就应该相对于 Content-Base 首部加以解释。如果没有提供 Content-Base 首部，就应该使用请求中的 URL。

类型　　实体首部

基本语法　Content-Location: (absoluteURL | relativeURL)

举例　　Content-Location: http://www.joes-hardware.com/index.html

---

Content-MD5

514 Content-MD5 首部是服务器用来对报文主体进行报文完整性检查的。只有原始服务器或发起请求的客户端可以在报文中插入 Content-MD5 首部。首部值是（可能需要编码的）报文主体的 MD5 摘要。[6]

通过这个首部的值可以端到端地检查数据，在检查传输过程中是否对数据进行了无意的修改时非常有用。不应该将其用于安全目的。

RFC 1864 更详细地定义了这个首部。

类型　　实体首部

注释　　根据 RFC 1864 的定义，MD5 摘要值是一个 Base-64（参见附录 E）或 128 位的 MD5 摘要。

基本语法　Content-MD5: md5-digest

举例　　Content-MD5: Q2h1Y2sgSW51ZwDIAXR5IQ==

Content-Range

请求传输某范围内的文档时，产生的结果由 Content-Range 首部给出。它提供了请求实体所在的原始实体内的位置（范围），还给出了整个实体的长度。

如果值为“*”，而不是整个实体的长度，就意味着发送响应时，长度未知。

更多有关 Content-Range 的内容请参见第 15 章。

类型　　实体首部

注释　　以 206 Partial Content 响应码进行响应的服务器，不能包含将“*”作为长度使用的 Content-Range 首部。

举例　　Content-Range: bytes 500-999 / 5400

注 6：MD5 摘要是在 RFC 1864 中定义的。

Content-Type

`Content-Type` 首部说明了报文中对象的媒体类型。

类型 实体首部

基本语法 `Content-Type: media-type`

举例 `Content-Type: text/html; charset=iso-latin-1` 515

Cookie

`Cookie` 首部是用于客户端识别和跟踪的扩展首部。第 11 章详细讨论了 `Cookie` 首部及其用法（还请参见 `Set-Cookie`）。

类型 扩展请求首部

举例 `Cookie: ink=IUOK164y59BC708378908CFF89OE5573998A115`

Cookie2

`Cookie2` 首部是用于客户端识别和跟踪的扩展首部。`Cookie2` 用于识别请求发起者能够理解哪种类型的 Cookie。在 RFC 2965 中对其进行了更加详细的定义。

第 11 章详细地讨论了 `Cookie2` 首部及其用法。

类型 扩展请求首部

举例 `Cookie2: $version="1"`

Date

`Date` 首部给出了报文创建的日期和时间。服务器响应中要包含这个首部，因为缓存在评估响应的新鲜度时，要用到这个服务器认定的报文创建时间和日期。对客户端来说，这个首部是可选的，但包含这个首部会更好。

类型 通用首部

基本语法 `Date: HTTP-date`

举例 `Date: Tue, 3 Oct 1997 02:15:31 GMT`

HTTP 有几种特定的日期格式。这种格式是在 RFC 822 中定义的，这是 HTTP/1.1 报文的优选格式。但在早期的 HTTP 规范中，没有明确说明日期的格式，因此服务器和客户端的实现者使用了一些其他格式，为了解决这些遗留问题仍然需要支持这些格式。你可能会碰到 RFC 850 中说明的那些日期格式，`asctime()` 系统调用产生的日期格式。下面是用这些格式所表示的上述日期：

```
Date: Tuesday, 03-Oct-97 02:15:31 GMT RFC 850 format
Date: Tue Oct 3 02:15:31 1997 asctime( ) format
```

大家都不喜欢 `asctime()` 格式，因为它表示的是本地时间，而且没有说明时区（比如，GMT）。总的来说，日期首部应该是 GMT 时间；但健壮的应用程序在处理

516 日期时，应该能够处理那些没有指定时区，或者包含了非 GMT 时间的 `Date` 值。

ETag

`ETag` 首部为报文中包含的实体提供了实体标记。实体标记实际上就是一种标识资源的方式。

第 15 章曾探讨过实体标记及其与资源之间的关系。

类型 实体首部

基本语法 `ETag: entity-tag`

举例 `ETag: "11e92a-457b-31345aa"`

`ETag: W/"11e92a-457b-3134b5aa"`

Expect

客户端通过 `Expect` 首部告知服务器它们需求某种行为。现在此首部与响应码 100 Continue 紧密相关（参见 3.4.1 节）。

如果服务器无法理解 `Expect` 首部的值，就应该以状态码 417 Expectation Failed 进行响应。

类型 请求首部

基本语法 `Expect: 1# ("100-continue" | expectation-extension)`

举例 `Expect: 100-continue`

Expires

`Expires` 首部给出了响应失效的日期和时间。这样，像浏览器这样的客户端就可以缓存一份副本，在这个时间到期之前，不用去询问服务器它是否有效了。

第 7 章曾讨论过 `Expires` 首部的用法——尤其是，它是如何与缓存关联，怎样与原始服务器进行响应再验证的。

类型 实体首部

基本语法 `Expires: HTTP-date`

举例 `Expires: Thu, 03 Oct 1997 17:15:00 GMT`

From

From 首部说明请求来自何方。其格式就是（RFC 1123 规定的）客户端用户的有效电子邮件地址。 517

使用 / 填充这个首部存在潜在的隐私问题。客户端实现者在请求报文中包含这个首部之前，应该通知用户，请他们作出选择。有些人会去收集不请自来的邮件报文中携带的电子邮件地址，这可能造成潜在的滥用。因此，未做声明就将此首部广播出去的实现者一定会非常懊悔，他们不得不向愤怒的用户说抱歉。

类型 请求首部

基本语法 `From: mailbox`

举例 `From: slurp@inktomi.com`

Host

客户端通过 Host 首部为服务器提供客户端想要访问的那台机器的因特网主机名和端口号。主机名和端口号来自客户端所请求的 URL。

只要服务器能够在同一台机器（即，在同一个 IP 地址）上提供多个不同的主机名，服务器就可以通过 Host 首部，根据主机名来区分不同的相对 URL。

类型 请求首部

注释 HTTP/1.1 客户端必须在所有请求中包含 Host 首部。所有的 HTTP/1.1 服务器都必须以 400 Bad Request 状态码去响应没有提供 Host 首部的客户端。

基本语法 `Host: host [":" port]`

举例 `Host: www.hotbot.com:80`

If-Modified-Since

If-Modified-Since 请求首部用来发起条件请求。客户端可以用 GET 方法去请求服务器上的资源，而响应则取决于客户端上次请求此资源之后，该资源是否被修改过。

如果对象未被修改过，服务器会回送一条 304 Not Modified 响应，而不会回送此资源。如果对象被修改过，服务器就会像对待非条件 GET 请求一样进行响应。第 7 章详细地讨论了条件请求。

类型 请求首部

基本语法 `If-Modified-Since: HTTP-date`

举例 `If-Modified-Since: Thu, 03 Oct 1997 17:15:00 GMT` 518

If-Match

与 If-Modified-Since 首部类似，If-Match 首部也可以用于发起条件请求。If-Match 请求使用的是实体标记，而不是日期。服务器将对比 If-Match 首部的实体标记与资源当前的实体标记，如果标记匹配，就将对象返回。

服务器应该用 If-Match 值“*”与某资源拥有的所有实体标记进行匹配。除非服务器上没有这个资源了，否则“*”总会与实体标记相匹配。

类型	请求首部
基本语法	`If-Match: ("*" | 1# entity-tag)`
举例	`If-Match: "11e92a-457b-31345aa"`

If-None-Match

与所有 If 首部一样，If-None-Match 首部可以用于发起条件请求。客户端为服务器提供一个实体标记列表，服务器将这些标记与它拥有的资源实体标记进行比较，只在都不匹配的时候才将资源返回。

这样缓存就可以只在资源已被修改的情况下才更新。通过 If-None-Match 首部，缓存可以用一条请求使它拥有的实体失效，同时在响应中接收新的实体。第 7 章曾讨论过条件请求。

类型	请求首部
基本语法	`If-None-Match: ("*" | 1# entity-tag)`
举例	`If-None-Match: "11e92a-457b-31345aa"`

If-Range

与所有 If 首部一样，If-Range 首部可以用于发起条件请求。应用程序拥有某范围内资源的副本，它要对范围进行再验证，如果范围无效的话，要获取新的资源，在这种情况下会使用这个首部。第 7 章详细讨论了条件请求。

类型	请求首部
519 **基本语法**	`If-Range: (HTTP-date | entity-tag)`
举例	`If-Range: Tue, 3 Oct 1997 02:15:31 GMT`
	`If-Range: "11e92a-457b-3134b5aa"`

If-Unmodified-Since

If-Unmodified-Since 和 If-Modified-Since 首部是一对“双胞胎”。在请求

中包含此首部就可以发起条件请求。服务器应该去查看首部的日期值，只有在从该首部提供的日期之后，对象都未被修改过，才会返回对象。第 7 章详细介绍了条件请求。

类型 请求首部

基本语法 `If-Unmodified-Since: HTTP-date`

举例 `If-Unmodified-Since: Thu, 03 Oct 1997 17:15:00 GMT`

Last-Modified

`Last-Modified` 首部试图提供这个实体最后一次被修改的相关信息。这个值可以说明很多事情。比如，资源通常都是一台服务器上的文件，因此 `Last-Modified` 值可能就是服务器的文件系统所提供的最后修改时间。另一方面，对于那些动态创建的资源（比如，由脚本创建的资源），`Last-Modified` 值可能就是创建响应的时间。

服务器要注意，`Last-Modified` 时间不应该是未来的时间。如果它比 `Date` 首部中要发送的值还迟，HTTP/1.1 服务器就会将 `Last-Modified` 时间重置。

类型 实体首部

基本语法 `Last-Modified: HTTP-date`

举例 `Last-Modified: Thu, 03 Oct 1997 17:15:00 GMT`

Location

服务器可以通过 `Location` 首部将客户端导向某个资源的地址，这个资源可能在客户端最后一次请求之后被移动过，也可能是在对请求的响应中创建的。

类型 响应首部

基本语法 `Location: absoluteURL`

举例 `Location: http://www.hotbot.com` 520

Max-Forwards

这个首部只能和 TRACE 方法一同使用，以指定请求所经过的代理或其他中间节点的最大数目。它的值是个整数。所有收到带此首部的 TRACE 请求的应用程序，在将请求转发出去之前都要将这个值减 1。

如果应用程序收到请求时，这个首部的值为零，就要向请求回应一条 200 OK 响应，并在实体的主体部分包含原始请求。如果 TRACE 请求中没有 `Max-Forwards` 首部，就假定没有转发最大次数的限制。

其他 HTTP 方法都应该忽略这个首部。更多有关 TRACE 方法的信息参见 3.3 节。

类型 请求首部

基本语法 `Max-Forwards: 1*DIGIT`

举例 `Max-Forwards: 5`

MIME-Version

MIME 是 HTTP 的近亲。尽管两者存在根本区别，但有些 HTTP 服务器确实构造了一些在 MIME 规范下同样有效的报文。在这种情况下，服务器可以提供 MIME 版本的首部。

尽管 HTTP/1.0 规范中提到过这个首部，但它从未写入官方规范。很多比较老的服务器会发送带有这个首部的报文，但这些报文通常都不是有效的 MIME 报文，这样会让人觉得这个首部令人迷惑且不可信。

类型 扩展的通用首部

基本语法 `MIME-Version: DIGIT "." DIGIT`

举例 `MIME-Version: 1.0`

Pragma

`Pragma` 首部用于随报文传送一些指令。这些指令几乎可以包含任何内容，但通常会用这些指令来控制缓存的行为。`Pragma` 首部的目标可以是接收这条报文的所有应用程序，因此代理和网关一定不能将其删除。

最常见的 `Pragma` 形式——`Pragma: no-cache` 是一个请求首部，通过它可以迫使缓存在有新鲜副本可用的情况下，向原始服务器请求文档或对其进行再验证。用户
521 点击重新加载 / 刷新按钮时，浏览器就会发出这个首部。很多服务器会将 `Pragma: no-cache` 作为响应首部发送（和 `Cache-Control:no-cache` 等价）。尽管这个首部得到了广泛的使用，但从技术上来说，并没有定义过其行为，不是所有的应用程序都支持 `Pragma` 响应首部。

第 7 章探讨了 `Pragma` 首部以及 HTTP/1.0 应用程序如何通过它来控制缓存。

类型 请求首部

基本语法 `Pragma: 1# pragma-directive`[7]

举例 `Pragma: no-cache`

注 7：规范中定义的唯一的一个 `Pragma` 指令就是 `no-cache`，但我们可能会碰到其他作为规范扩展而定义的 `Pragma` 首部。

Proxy-Authenticate

Proxy-Authenticate 首部的功能类似于 WWW-Authenticate 首部。代理会这个首部来质询发送请求的应用程序，要求其对自身进行认证。第 14 章详细讨论了这个质询 / 响应过程和 HTTP 的其他安全机制。

如果一台 HTTP/1.1 代理服务器发送了一条 407 Proxy Authentication Required 响应，就必须包含 Proxy-Authenticate 首部。

代理和网关在解释所有的 Proxy 首部时要特别小心。通常它们都是逐跳首部，只适用于当前的连接。比如，Proxy-Authenticate 首部会要求对当前的连接进行认证。

类型　响应首部

基本语法　`Proxy-Authenticate: challenge`

举例　`Proxy-Authenticate: Basic realm="Super Secret Corporate FinancialDocuments"`

Proxy-Authorization

Proxy-Authorization 首部的功能与 Proxy-Authorization 首部类似。客户端应用程序可以用它来响应 Proxy-Authenticate 质询。更多有关质询 / 响应安全机制工作原理的内容参见第 14 章。

类型　请求首部

基本语法　`Proxy-Authorization: credentials`

举例　`Proxy-Authorization: Basic YnJpYW4tdG90dHk6T3ch` 522

Proxy-Connection

Proxy-Connection 首部的语义与 HTTP/1.0 Connection 首部类似。在客户端和代理之间可以用它来指定与连接（主要是 keep-alive 连接）有关的选项。[8] 它并不是一个标准的首部，标准委员会把它当作一个临时首部。但它得到了浏览器和代理的广泛使用。

浏览器实现者创建了 Proxy-Connection 首部，来解决客户端发送的 HTTP/1.0 Connection 首部被哑代理盲转发的问题。收到被盲转发的 Connection 首部的服务器会将客户端连接的功能与代理连接的功能混淆起来。

注 8：更多有关 keep-alive 和持久连接的内容参见第 14 章。

客户端知道要经过代理传输时，就会发送 Proxy-Connection 首部，而不是 Connection 首部。服务器如果无法识别 Proxy-Connection 首部，就会将其忽略，这样，对首部进行盲转发的哑代理就不会带来任何问题了。

如果在从客户端到服务器的路径上有多个代理，这种解决方法就会有问题。如果第一个代理将首部盲转发给第二个能够理解它的代理，那么第二个代理就会像服务器看到 Connection 首部一样，无法理解。

这是 HTTP 工作组的解决方案存在的问题——他们将其作为一种黑客工具，可以解决单个代理的问题，但无法解决更大的问题。尽管如此，这种方式确实能够处理一些比较常见的情况，而且由于网景的 Navigator 和微软的 Internet Explorer 的较老版本都实现了这个首部，因而代理的实现者也需要对其进行处理，更多信息参见第 4 章。

类型 通用首部

基本语法 `Proxy-Connection: 1# (connection-token)`

举例 `Proxy-Connection: close`

Public

服务器可以用 Public 首部告知客户端它支持哪些方法。今后客户端发起的请求就可以使用这些方法了。代理收到服务器发出的带有 Public 首部的响应时，要特别小心。这个首部说明的是服务器支持的方法，而不是代理的，因此代理在将响应发送给客户端之前，要对首部的方法列表加以编辑，或者将此首部删除。

523 **类型** 响应首部

注释 RFC 2616 中没有定义这个首部。它是之前在 HTTP/1.1 规范的早期草案 RFC 2068 中定义的，而官方规范已经将其删除了。

基本语法 `Public: 1# HTTP-method`

举例 `Public: OPTIONS, GET, HEAD, TRACE, POST`

Range

在请求某实体的部分内容中会用到 Range 首部。它的值说明了报文所包含实体的范围。

请求某范围内的文档可以更有效地对大型对象发出请求（分段对其发出请求），或者更有效地从传输错误中恢复（允许客户端请求没有完成的那部分资源）。第 15 章详细说明了范围请求和能实现范围请求的首部。

类型　　实体首部

举例　　`Range: bytes=500-1500`

`Referer`

在客户端请求中插入 `Referer` 首部，可以使服务器知道客户端是从哪里获得其请求的 URL。这是一种对服务器有益的自愿行为，这样服务器就可以更好地记录请求，或执行其他任务了。Referer 的拼写错误要回溯到 HTTP 的早期，令世界各地以英语为母语的文字编辑们万分沮丧。

浏览器所做的工作相当简单。如果在主页 A 上点击一个链接，进入主页 B，浏览器就会在请求中插入一个带有值 A 的 `Referer` 首部。只有在你点击链接的时候，浏览器才会插入 `Referer` 首部；自己输入的 URL 中不会包含 `Referer` 首部。

因为有些页面是私有的，所以这个首部会有隐私问题。尽管有些只是毫无根据的猜想，但 Web 服务器及其管理者确实可以通过这个首部看到你来自何方，这样他们就能更好地追踪你的浏览行为了。因此，HTTP/1.1 规范建议应用程序编写者让用户来选择是否传输这个首部。

类型　　请求首部

基本语法　　`Referer: (absoluteURL | relativeURL)`

举例　　`Referer: http://www.inktomi.com/index.html` 524

`Retry-After`

服务器可以用 `Retry-After` 首部告知客户端什么时候重新发送某资源的请求。这个首部可以与 503 Service Unavailable（服务不可用）状态码配合使用，给出客户端可以重试其请求的具体日期和时间（或者秒数）。

服务器还可以在将客户端重定向到资源时，通过这个首部通知客户端在对重定向的资源发送请求之前需要等待的时间。[9] 对那些正在创建动态资源的服务器来说，这个首部是非常有用的，服务器可以通过它将客户端重定向到新创建的资源，并给出了资源创建所需的时间。

类型　　响应首部

基本语法　　`Retry-After: (HTTP-date | delta-seconds)`

举例　　`Retry-After: Tue, 3 Oct 1997 02:15:31 GMT`

　　　　`Retry-After: 120`

注 9：更多有关服务器重定向响应的信息参见表 3-8。

Server

Server 首部与 User-Agent 首部类似。它为服务器提供了一种向客户端标识自己的方式。它的值就是服务器名字和一个可选的服务器注释。

Server 首部是用来识别服务器软件的，而且包含了与软件有关的附加注释，所以其格式比较随意。如果编写的软件与服务器标识自己的方式有关，就应该测试服务器软件，看看它会发回什么内容，因为这些标记会随软件及其发布版本的不同而有所不同。

像 User-Agent 首部一样，如果较老的代理或网关在 Server 首部中插入了相当于 Via 首部的内容，千万不要感到吃惊。

类型 响应首部

基本语法 `Server: 1* (product | comment)`

举例
```
Server: Microsoft-Internet-Information-Server/1.0
Server: Websitepro/1.1f (s/n wpo-07d0)
Server: apache/1.2b6 via proxy gateway CERN-HTTPD/3.0
libwww/2.13
```

Set-Cookie

525 Set-Cookie 首部是 Cookie 首部的搭档。第 11 章介绍了这个首部的用法。

类型 扩展响应首部

基本语法 `Set-Cookie: command`

举例
```
Set-Cookie: lastorder=00183; path=/orders
Set-Cookie: private_id=519; secure
```

Set-Cookie2

Set-Cookie2 首部是对 Set-Cookie 首部的扩展。第 11 章详细了探讨了这个首部的用法。

类型 扩展响应首部

基本语法 `Set-Cookie2: command`

举例
```
Set-Cookie2: ID="29046"; Domain=".joes-hardware.com"
Set-Cookie2: color=blue
```

TE

TE 首部的名字起得不太好（本应该将其命名为 Accept-Transfer-Encoding），

它的功能与 Accept-Encoding 首部类似，但它是用于传输编码的。TE 首部还可以用来说明客户端能否处理位于分块编码的响应拖挂中的首部。更多有关 TE 首部、分块编码和拖挂的内容参见第 15 章。

类型 请求首部

注释 如果这个值为空，就只接受分块传输编码。特定标记“trailers”说明分块响应中可以接受 Trailer 首部。

基本语法
```
TE: # (transfer-codings)
transfer-codings= "trailers" | (transfer-extension [accept-params])
```

举例
```
TE:
TE: chunked
```

Trailer

Trailer 首部用于说明报文拖挂中提供了哪些首部。第 15 章详细说明了分块编码和拖挂。

类型 通用首部 526

基本语法 `Trailer: 1#field-name`

举例 `Trailer: Content-Length`

Title

Title 首部不像人们所期望的那样，会给出实体标题的规范化首部。这个首部是早期 HTTP/1.0 扩展的一部分，主要用于 HTML 页面，这些 HTML 页面有着服务器可以使用的明确的标题标记。但即使不是大部分，也有很多 Web 媒体类型没有便捷的标题解析手段，这个标题的用处有限。因此，尽管网络上一些比较老的服务器仍然在忠实地发送这个首部，但它从未成为官方规范。

类型 响应首部

注释 RFC 2616 中没有定义 Title 首部。最早是在 HTTP/1.0 草案（http://www.w3.org/Protocols/HTTP/HTTP2.html）中定义的，但之后就从官方规范中删除了。

基本语法 `Title: document-title`

举例 `Title: CNN Interactive`

Transfer-Encoding

如果要通过某些编码来安全地传送 HTTP 报文主体，报文中就要包含 Transfer-

Encoding 首部。它的值是一个对报文主体执行过的编码的列表。如果进行了多种编码，就将其按序排列。

Transfer-Encoding 首部与 Content-Encoding 首部不同，因为服务器或其他中间应用程序是通过执行 Transfer-Encoding 对要传输的报文进行编码的。

第 15 章介绍过传输编码。

类型　通用首部

基本语法　`Transfer-Encoding: 1# transfer-coding`

举例　`Transfer-Encoding: chunked`

UA-(CPU, Disp, OS, Color, Pixels)

这些 User-Agent 首部是非标准的，现在也不常见了。它们提供了客户端机器的相
527 关信息，以便服务器更好地进行内容选择。比如，如果服务器知道用户机器只有一
个 8 位彩色显示器，服务器就可以选择适合那类显示器的图片了。

有些首部给出了与客户端相关的信息，不使用这些首部就无法获知这些信息。所有这样的首部都有一些安全方面的隐患（更多信息参见第 14 章）。

类型　扩展请求首部

注释　RFC 2616 没有定义这些首部，而且不推荐使用这些首部。

基本语法

```
"UA" "-" ("CPU" | "Disp" | "OS" | "Color" | "Pixels")
":" machine-value
machine-value = (cpu | screensize | os-name |display-
color-depth)
```

举例

UA-CPU: ×86	客户端机器的 CPU
UA-Disp: 640, 480, 8	客户端显示器的尺寸和色彩深度
UA-OS: Windows 95	客户端机器的操作系统
UA-Color: color8	客户端显示器的色彩深度
UA-Pixels: 640×480	客户端显示器的尺寸

Upgrade

Upgrade 首部为报文发送者提供了一种手段，使其指定另一种可能完全不同协议并将此意愿向外广播。比如，HTTP/1.1 客户端可以向服务器发送一条 HTTP/1.0 请求，其中包含了值为“HTTP/1.1”的 Update 首部，这样客户端就可以测试一下服务器是否也使用 HTTP/1.1 了。

如果服务器也可以使用 HTTP/1.1，就可以发送一条适当的响应，让客户端知道可以使用新的协议。这样就提供了一种切换使用其他协议的有效方式。现在大部分服务器都只兼容 HTTP/1.0，通过这种策略，在判定服务器确实能够使用 HTTP/1.1 之前，客户端就不会用很多的 HTTP/1.1 首部骚扰服务器了。

服务器发送 101 Switching Protocols 响应时，必须包含这个首部。

类型 通用首部

基本语法 `Upgrade: 1# protocol`

举例 `Upgrade: HTTP/2.0`

User-Agent

客户端应用程序用 `User-Agent` 首部来标识其类型，与服务器的 `Server` 首部类似。它的值就是应用程序的名称，可能还会有一个描述性注释。 528

这个首部的格式比较随意。它的值会随客户端应用程序和发布版本的不同而有所不同。有时这个首部甚至会包含一些有关客户端机器的信息。

与 `Server` 首部一样，如果较老的代理或网关应用程序在 `User-Agent` 首部中插入了与 `Via` 首部等效的内容，请不要感到惊奇。

类型 请求首部

基本语法 `User-Agent: 1* (product | comment)`

举例 `User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)`

Vary

服务器通过 `Vary` 首部来通知客户端，在服务器端的协商中会使用哪些来自客户端请求的首部。[10] 它的值是一个首部列表，服务器会去查看这些首部，以确定将什么内容作为响应发回给客户端。

根据客户端 Web 浏览器特性来发送特定 HTML 页面的服务器就是一例。为某个 URL 发送这些特定页面的服务器会包含一个 `Vary` 首部，以说明它是查看了请求的 `User-Agent` 首部之后，才决定发送什么内容作为响应的。

代理缓存也会使用 `Vary` 首部。更多有关 `Vary` 首部与已缓存的 HTTP 响应关联方式的信息参见第 7 章。

注 10：更多与内容协商有关的内容参见第 17 章。

类型 响应首部

基本语法 `Vary: ("*" | 1# field-name)`

举例 `Vary: User-Agent`

Via

Via 首部用于在报文经过代理和网关时对其进行跟踪。这是一个信息首部，通过它可以看出哪些应用程序在对请求和响应进行处理。

报文在向客户端或服务器传输的途中经过某个 HTTP 应用程序时，这个应用程序可以通过 Via 首部对通过它传输的报文进行标记。这是个 HTTP/1.1 首部，而很多较老的应用程序会在请求和响应的 User-Agent 或 Server 首部插入类似 Via 的字符串。

如果报文是通过多个中间应用程序传输的，那么每个应用程序都会向其 Via 字符串
529 中附加一些内容。必须通过 HTTP/1.1 代理和网关来插入 Via 首部。

类型 通用首部

基本语法 `Via: 1# (received-protocol received-by [comment])`[11]

举例 `Via: 1.1 joes-hardware.com ( Joes-Server/1.0)`

上面这个例子说明报文是通过运行在机器 joes-hardware.com 上的 Joes 的服务器软件 1.0 版传输的。Via 首部的格式应该如下所示：

```
HTTP-Version machine-hostname (Application-Name-Version)
```

Warning

Warning 首部可以给出更多与请求过程中所发生情况有关的信息。它为服务器提供了一种手段，可以发送除状态码或原因短语之外的其他信息。HTTP/1.1 规范中定义了以下几种警告代码。

- 101 响应过时了

当知道一条响应报文已过期时（比如，原始服务器无法进行再验证时），就必须包含这条警告信息。

- 111 再验证失败

如果缓存试图与原始服务器进行响应再验证，但由于缓存无法抵达原始服务器造成了再验证失败，那就必须在发给客户端的响应中包含这条警告信息。

注 11：完整的 Via 头部语法参见 HTTP/1.1 规范。

- 112 断开连接操作

通知性警告信息。如果缓存到网络的连接被删除了就应该使用此警告信息。

- 113 试探性过期

如果新鲜性试探过期时间大于 24 小时，而且返回的响应使用期大于 24 小时，缓存中就必须包含这条警告信息。

- 199 杂项警告

收到这条警告的系统不能使用任何自动响应。报文中可能，而且很可能应该包含一个主体，其中携带了为用户提供的额外信息。

- 214 使用了转换

如果中间应用程序执行了任何会改变响应内容编码的转换，就必须由任意一个中间应用程序（比如代理）来添加这条警告。

- 299 持久杂项警告

接收这条警告的系统不能进行任何自动的回应。错误中可能包含一个主体部分，它为用户提供了更多的信息。

类型　响应首部

基本语法　`Warning: 1# warning-value`

举例　`Warning: 113`

530

WWW-Authenticate

`WWW-Authenticate` 首部用于 401 Unauthorized 响应，向客户端发布一个质询认证方案。第 14 章深入讨论了 `WWW-Authenticate` 首部及其在 HTTP 基本质询 / 响应认证系统中的使用方法。

类型　响应首部

基本语法　`WWW-Authenticate: 1# challenge`

举例　`WWW-Authenticate: Basic realm="Your Private Travel Profile"`

X-Cache

X 开头的都是扩展首部。Squid 用 `X-Cache` 首部来通知客户端某个资源是否可用。

类型　扩展响应首部

举例　`X-Cache: HIT`

X-Forwarded-For

很多代理服务器（比如，Squid）会用这个首部来说明某条请求都被转发给了谁。与前面提到的 `Client-ip` 首部类似，这个请求首部说明了请求是从哪个地址发出的。

类型　扩展请求首部

基本语法　`X-Forwarded-For: addr`

举例　`X-Forwarded-For: 64.95.76.161`

X-Pad

这个首部用来解决某些浏览器中与响应首部长度有关的 bug。它在响应报文的首部
531 填充了一些字节，以解决这个 bug。

类型　扩展通用首部

基本语法　`X-Pad: pad-text`

举例　`X-Pad: bogosity`

X-Serial-Number

`X-Serial-Number` 首部是个扩展首部。某些较老的 HTTP 应用程序会用它向 HTTP 报文中插入许可软件的序列号。

它基本上已经没什么用处了，它只是作为 X 开头的首部的一个示例列在这里。

类型　扩展通用首部

基本语法　`X-Serial-Number: serialno`

532 **举例**　`X-Serial-Number: 010014056`

附录D

MIME类型

MIME 媒体类型（简称为 MIME 类型）是描述报文实体主体内容的一些标准化名称（比如，text/html、image/jpeg）。本附录说明了 MIME 类型是怎样工作的、如何注册新的名字以及到哪里去查找更多的相关信息。

本附录还包含了 10 张便捷表格，详细描述了从全球众多资源中搜集来的数百种 MIME 类型。这可能是有史以来最详细的 MIME 类型列表。希望这些表格会对你有所帮助。

此附录的主要内容有：

- 在 D.1 节列出主要的参考资料；
- 在 D.2 节解释 MIME 类型的结构；
- 在 D.3 节说明了如何注册 MIME 类型；
- 附加的 10 张表格让你更便捷地查找 MIME 类型。

本附录包含了下列 MIME 类型表：

- application/*——表 D-3
- audio/*——表 D-4
- chemical/*——表 D-5
- image/*——表 D-6
- message/*——表 D-7
- model/*——表 D-8
- multipart/*——表 D-9
- text/*——表 D-10
533 • video/*——表 D-11
- 其他——表 D-12

D.1 背景知识

MIME 类型最初是为多媒体电子邮件而开发的，但目前在 HTTP 和其他几种需要描述数据对象格式和用途的协议中也使用了 MIME 类型。

MIME 主要由下列 5 份文档定义。

- RFC 2045，“MIME: Format of Internet Message Bodies”（“MIME：因特网报文主体的格式”）
  描述了 MIME 报文结构的概况，并介绍了 HTTP 借用的 `Content-Type` 首部。

- RFC 2046，“MIME: Media Types”（“MIME：媒体类型”）

介绍了 MIME 类型及其结构。

- RFC 2047，“MIME: Message Header Extensions for Non-ASCII Text”（“MIME：非 ASCII 文本的报文首部扩展”）
  定义了一些在首部包含非 ASCII 字符的方式。
- RFC 2048，“MIME: Registration Procedures”（“MIME：注册过程”）
  定义了如何向因特网号码分配机构（“Internet Assigned Numbers Authority，IANA”）注册 MIME 值。
- RFC 2049，“MIME: Conformance Criteria and Examples”（“MIME：一致性标准及实例”）
  详细介绍了一致性规则，并提供了一些实例。

根据 HTTP 的目标，我们最感兴趣的文档是 RFC 2046 和 RFC 2048。

D.2　MIME类型结构

每种 MIME 媒体类型都包含类型、子类型和可选参数的列表。类型和子类型由一个斜杠分隔，如果有可选参数的话，则以分号开始。在 HTTP 中，MIME 媒体类型被广泛用于 `Content-Type` 和 `Accept` 首部。下面是几个例子：

```
Content-Type: video/quicktime
Content-Type: text/html; charset="iso-8859-6"
Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
Accept: image/gif
```

D.2.1　离散类型

MIME 类型可以直接用于描述对象类型，也可以用于描述其他对象类型的集合或类型包。如果直接用 MIME 类型来描述某个对象类型，它就是一种离散类型（discrete type）。其中包括文本文件、视频和应用程序特有的文件格式。

D.2.2　复合类型

如果 MIME 类型描述的是其他内容的集合或封装包，这种 MIME 类型就被称为复合类型（composite type）。复合类型描述的是封装包的格式。将封装包打开时，其中包含的每个对象都会有其各自的类型。 534

D.2.3　多部分类型

多部分媒体类型是复合类型。多部分对象包含多个组件类型。下面是一个多部分 / 混合内容实例，每个组件都有自己的 MIME 类型：

```
Content-Type: multipart/mixed; boundary=unique-boundary-1

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

Hi there, I'm some boring ASCII text...

--unique-boundary-1
Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic

    ... 8000 Hz single-channel mu-law-format
        audio data goes here ...

--unique-boundary-2
Content-Type: image/jpeg

    ... image data goes here ...

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

This is <bold><italic>enriched.</italic></bold>
<smaller>as defined in RFC 1896</smaller>

Isn't it <bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

    ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

535

D.2.4　语法

如前所述，MIME 类型由主类型、子类型和可选参数的列表组成。

主类型可以是预定义类型、IETF 定义的扩展标记，或者（以“x-”开头的）实验性标记。表 D-1 列出了一些常见的主类型。

表D-1　常见的主MIME类型

类　　型	描　　述
application	应用程序特有的内容格式（离散类型）
audio	音频格式（离散类型）
chemical	化学数据集（离散 IETF 扩展类型）
image	图片格式（离散类型）
message	报文格式（复合类型）
model	三维模型格式（离散 IETF 扩展类型）
multipart	多部分对象集合（复合类型）
text	文本格式（离散类型）
video	视频电影格式（离散类型）

子类型可以是主类型（比如，“text/text”）、IANA 注册的子类型，或者是（以“x-”开头的）实验性扩展标记。

类型和子类型都是由 US-ASCII 字符的一个子集构成的。空格和某些保留分组以及标点符号称为“tspecials”，它们是控制字符，不能用于类型和子类型名。

RFC 2046 定义的语法如下所示：

```
TYPE := "application" | "audio" | "image" | "message" | "multipart" |
        "text" | "video" | IETF-TOKEN | X-TOKEN
SUBTYPE := IANA-SUBTOKEN | IETF-TOKEN | X-TOKEN

IETF-TOKEN := <extension token with RFC and registered with IANA>
IANA-SUBTOKEN := <extension token registered with IANA>
X-TOKEN := <"X-" or "x-" prefix, followed by any token>

PARAMETER := TOKEN "=" VALUE
VALUE := TOKEN / QUOTED-STRING
TOKEN := 1*<any (US-ASCII) CHAR except SPACE, CTLs, or TSPECIALS>
TSPECIALS := "(" | ")" | "<" | ">" | "@" |
             "," | ";" | ":" | "\" | <"> |
             "/" | "[" | "]" | "?" | "="
```
536

D.3　在IANA注册MIME类型

RFC 2048 描述了 MIME 媒体类型的注册过程。使用注册过程的目的不仅能简化注册新媒体类型的过程，而且还能提供一些完整性检测，以确保新类型是经过深思熟虑后提出的。

D.3.1 注册树

MIME 类型标记被分成了 4 类，称为“注册树”，每一类都有自己的注册规则。表 D-2 描述了这 4 棵树——IETF、厂商、个人和实验性的树。

表D-2 4种MIME媒体类型注册树

注 册 树	举 例	描 述
IETF	text/html（HTML 文本）	IETF 树用于那些对因特网来说具有普遍意义的类型。新的 IETF 树媒体类型要由因特网工程指导组（Internet Engineering Steering Group，IESG） 审批，并且要有一个附属的标准追踪 RFC。 IETF 树类型标记中没有句点（.）
厂商（vnd.）	image/vnd.fpx （柯达的 FlashPix 图片 ）	厂商树用于可商用产品的媒体类型。鼓励大家进行新厂商类型的公开评审，但并不强制这么做。 厂商树类型以 vnd. 开头
个人 / 不重要的（prs.）	image/prs.btif （美国国家银行使用的内部检查管理格式）	私有、个人或不重要的媒体类型可以注册在个人树中。这些媒体类型不应该进行商业化发布。 个人数类型以 prs. 开头
实验性（x- 或 x.）	application/x-tar （Unix 的 tar 档案）	实验树用于未注册或实验性的媒体类型。由于注册新的厂商或个人媒体类型的过程相对简单，所以不应该广泛地发布使用 x- 类型的软件。 实验树类型以 x. 或 x- 开头

D.3.2 注册过程

关于 MIME 媒体类型的注册过程请仔细阅读 RFC 2048。

基本的注册过程并不是官方的标准过程，只是一个管理过程，目的是在尽短时间内，就能通过 IANA 对新类型的完整性检查，将其记录在注册表中。这个过程遵循下列步骤。

（1）将媒体类型提交给 IANA 审阅。

向 ietf-types@iana.org 邮件列表发送一封媒体类型注册建议书，审阅期为两周。发布公告，征求公众对名字、互操作性和安全问题的反馈意见。在注册完成之
537 前，都可以使用 RFC 2045 中指定的前缀 x-。

（2）IESG 审批（仅对 IETF 树而言）

如果要将媒体类型注册到 IETF 树中去，就必须提交给 IESG 审批，而且必须有一个附加的标准追踪 RFC。

（3）IANA 注册

只要媒体类型达到了审批要求，作者就可以通过例 D-1 中的电子邮件模板向 IANA 提交注册请求，并将注册信息发送到 ietf-types@iana.org。IANA 会注册媒体类型，在 http://www.isi.edu/in-notes/iana/assignments/media-types/ 上向公众提供该媒体类型应用程序。

D3.3 注册规则

只有在响应 IESG 批准某指定注册申请的通知时，IANA 才能在 IETF 树中注册媒体类型。

只要满足下面这些条件，IANA 就会自动注册厂商和个人类型，不需要进行任何正式的审查。

（1）媒体类型一定要像实际的媒体格式一样工作。像传输编码或字符集那样工作的类型是不能注册为媒体类型的。

（2）所有媒体类型都要有适当的类型和子类型名。所有类型名都要由标准追踪 RFC 定义。所有子类型名都必须是唯一的，必须与那类名称的 MIME 语法相符，而且必须包含恰当的树前缀。

（3）个人树类型必须提供格式规范或指向格式规范的指针。

（4）不要忽略安全问题。所有开发因特网软件的人都要为防范安全漏洞作出贡献。

D.3.4 注册模板

实际的 IANA 注册是通过电子邮件完成的。可以用例 D-1 中显示的模板来完成注册表格，并将其发送到 ietf-types@iana.org。[1]

例 D-1 IANA 的 MIME 注册电子邮件模板

```
To: ietf-types@iana.org
Subject: Registration of MIME media type XXX/YYY

MIME media type name:
```

538

注 1：这个格式没有很严格的结构，稍做调整并不会影响信息阅读，但由机器处理则比较困难。这就是很难找到一份可读的、经过良好组织的 MIME 类型小结的原因之一，这也是我们在本附录末尾创建那些表格的原因。

```
MIME subtype name:

Required parameters:

Optional parameters:

Encoding considerations:

Security considerations:

Interoperability considerations:

Published specification:

Applications which use this media type:

Additional information:

    Magic number(s):
    File extension(s):
    Macintosh File Type Code(s):

Person & email address to contact for further information:

Intended usage:

(One of COMMON, LIMITED USE or OBSOLETE)

Author/Change controller:

(Any other information that the author deems interesting may be added
below this line.)
```

D.3.5 MIME媒体类型注册

可以通过 IANA 的网站（http://www.iana.org）访问那些已提交的表格。编写本书时，实际存储 MIME 媒体类型的数据库位于 http://www.isi.edu/in-notes/iana/assignments/media-types/ 的 ISI Web 服务器上。

媒体类型存储在一棵目录树中，以主类型和子类型进行结构划分，每种媒体类型都有一个叶子文件。每个文件中都包含了电子邮件提交信息。但是，每个人所完成的注册模板都略有不同，因此，不同人提交的信息质量和格式都有所不同。（在本附录的表格中，我们试着填补了注册用户忽略的部分。）

D.4 MIME类型表

本节在 10 张表格中对数百种 MIME 类型进行了总结。每张表格都列出了一个特定
539 主类型（image、text 等）中的 MIME 媒体类型。

这些信息是从很多地方搜集来的，包括 IANA 的媒体类型注册登记处、Apache 的 mime.types 文件，以及各种各样的因特网 Web 页面。我们花了好多天来提炼数据，填补漏洞，并在其中包含了交叉参考文献中的描述性总结，使数据的可用性更高。

这可能是有史以来最详细的 MIME 类型清单列表了。希望对你有所帮助！

D.4.1 application/*

表 D-3 描述了很多应用程序特有的 MIME 媒体类型。

表D-3 Application MIME类型

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/activemessage	支持主动邮件群件系统		*Readings in Groupware and Computer-Supported CooperativeWork*,“Active Mail: A Framework for Integrated Groupware Applications”，Ronald M. Baecker，Morgan Kaufmann 出版社，ISBN1558602410
application/andrew-inset	支持用安德鲁工具集创建多媒体内容	ez	*Multimedia Applications Development with the Andrew Toolkit*,Nathaniel S. Borenstein，Prentice Hall 出版社，AS-IN 0130366331 nsb@bellcore.com
application/applefile	在允许对非特定用户数据进行一般性访问的同时，允许对带有 Apple/Macintosh 特有信息的数据进行基于 MIME 的传输		RFC 1740
application/atomicmail	ATOMICMAIL 是贝尔通信研究所的一个实验型研究项目，设计用来在电子邮件报文中包含一些在阅读邮件时会执行的程序。对 safe-tcl 的支持会很快使 ATOMICMAIL 过时		“ATOMICMAIL Language Reference Manual”，Nathaniel S. Borenstein，贝尔科技备忘录，TM ARH-018429
application/batch-SMTP	定义了一个 MIME 内容类型，可以用隧道方式通过任意能够进行 MIME 传输的路径实现 ESMTP 邮件事务		RFC 2442

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/beep+xml	支持名为BEEP的交互协议。BEEP支持在对等实体之间对MIME报文进行同步且独立的交互，这些报文通常都是XML结构的文本		RFC 3080
application/cals-1840	支持对美国国防部数字数据进行MIME电子邮件交换，早期这些数据是MILSTD-1840定义的由磁带实现数据交换的		RFC1895
application/commonground	Common Ground是个电子文档交换及发布程序，允许用户创建一些文档，这些文档无需在系统中创建任何应用程序或字体，所有人都可以查看、搜索和打印		Nick Gault No Hands软件 ngault@nohands.com
application/cybercash	支持通过CyberCash协议进行信用卡支付。用户开始付费时，商家会向客户发送一条报文，作为MIME类型application/cybercash的报文主体使用		RFC 1898
application/dca-rft	IBM文档内容结构		"IBM Document Content Architec-ture/Revisable Form Text Reference"，文档号SC23-0758-1，国际商用机器公司（International Business Machines，IBM）
application/dec-dx	DEC的文档传输格式		"Digital Document Transmission (DX) Technical Notebook"，文档号EJ29141-86，数字设备公司（DEC，Digital Equipment Corporation）
application/dvcs	支持DVCS（Data Validation and Certification Server，数据验证及证书服务器）所使用的协议，该服务器在公共密钥安全架构中作为第三方受信站点使用的		RFC 3029
application/EDI-Consent	支持通过EDI（electronic data interchange，电子数据交换）用非标准的规范进行双边贸易		http://www.isi.edu/in-notes/iana/assignments/media-types/application/EDI-Consent

540

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/ EDI-X12	支持通过 EDI 用 ASC X12 EDI 规范进行双边贸易		http://www.isi.edu/in-notes/iana/ assignments/media-types/application/ EDI-X12
application/ EDIFACT	支持通过 EDI 用 EDIFACT 规范进行双边贸易		http://www.isi.edu/in-notes/iana/ assignments/media-types/application/ EDIFACT
application/ eshop	未知		Steve Katz 系统结构商店 steve_katz@eshop.com
application/ font-tdpfr	定义了包含一组字形的 PFR（Portable Font Resource，便携式字体资源），每个字形都与一个字符码相关		RFC 3073
application/http	用于封装由一条或多条 HTTP 请求或响应报文（不混合）构成的管道		RFC 2616
application/ hyperstudio	支持 HyperStudio 教学超媒体文件的传输	stk	http://www.hyperstudio.com
application/iges	CAD 模型交换的常用格式		“ANS/US PRO/IPO-100”U.S. Product Data Association2722 Merrilee Drive, Suite 200 Fairfax, VA 22031-4499
application/ index application/ index.cmd application/ index.obj application/ index.response application/ index.vnd	支持 CIP（Common Indexing Protocol，公共索引协议）。CIP 是对 Whois++ 目录服务的发展，用于在服务器间传送索引信息，以便通过分布式数据库系统来重定向或复制请求		RFC 2652、RFC 2651、RFC1913 和 RFC1914
application/iotp	支持在 HTTP 上发送 IOTP（Internet Open Trading Protocol，因特网开放贸易协议）报文		RFC 2935

541

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/ipp	支持在 HTTP 上使用 IPP（Internet Printing Protocol，因特网打印协议）		RFC 2910
application/mac-binhex40	将 8 位字节的字符串编码为 7 位字节的字符串，这样对某些应用程序来说更安全一些（没有 6 位的 Base-64 编码安全）	hqx	RFC 1341
application/mac-compactpro	来自 Apache 的 mime.types	cpt	
application/macwriteii	Claris MacWrite II		
application/marc	MARC 对象是机读编目记录——书目及相关信息的表示和通信标准	mrc	RFC 2220
application/mathematica application/mathematica-old	支持 Mathematica 和 MathReader 数值分析软件	nb、ma、mb	*The Mathematica Book*，Stephen Wolfram，剑桥大学出版社，ISBN 0521643147
application/msword	微软的 Word MIME 类型	doc	
application/news-message-id			RFC 822（报文 ID）、RFC1036（新闻应用程序）以及 RFC977（NNTP）
application/news-transmission	允许通过电子邮件或其他传输方式传送新闻文章		RFC 1036
application/ocsp-request	支持 OCSP（Online Certificate Status Protocol，在线证书状态协议），此协议提供了一种无需本地证书撤销列表即可查看数字证书有效性的方法	orq	RFC 2560
application/ocsp-response	同上	ors	RFC 2560
application/octet-stream	未分类的二进制数据	bin、dms、lha、lzh、exe、class	RFC 1341

542

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/oda	根　据 ODA（Office Document Architecture，办公文档结构）标准， 用 ODIF（Office Document Interchange Format，办公文档交互格式）表示法对信息进行编码。`Content-Type` 行也应该像：`Content-Type:application/oda;profile=Q112` 这样指定用来说明 DAP（document application profile， 文档应用外观）的属性 / 值对	oda	RFC 1341 ISO 8613；“Information Processing: Text and Office System; Office Document Architecture (ODA) and Interchange Format (ODIF)”， 第 1-8 部分，1989 年
application/parityfec	RTP 数据流的前向纠错奇偶码		RFC 3009
application/pdf	Adobe PDF 文件	pdf	参见 *Portable Document Format Reference Manual*，Adobe System, Inc. Addison Wesley，ISBN0201626284
application/pgp-encrypted	PGP 加密数据		RFC 2015
application/pgp-keys	PGP 公共密钥块		RFC 2015
application/pgp-signature	PGP 加密签名		RFC 2015
application/pkcs10	公共密钥加密系统 #10——传输 PKCS #10 证书请求时，主体类型必须用 application/pkcs10	p10	RFC 2311
application/pkcs7-mime	公共密钥加密系统 #7——这种类型用于传送包括 envelopedData 和 signedData 在内的几种类型的 PKCS #7 对象	p7m	RFC 2311
application/pkcs7-signature	公共密钥加密系统 #7——这种类型总是包含一个 signedData 类型的 PKCS #7 对象	p7s	RFC 2311
application/pkix-cert	传输 X.509 证书	cer	RFC 2585
application/pkix-crl	传输 X.509 证书撤销列表	crl	RFC 2585

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/pkixcmp	X.509 公共密钥基础设施证书管理协议所使用的报文格式	pki	RFC 2510
application/postscript	Adobe PostScript 图像文件（程序）	ai、ps、eps	RFC 2046
application/prs.alvestrand.titrax-sheet	Harald T. Alvestrand 的“TimeTracker”程序		http://domen.uninett.no/~hta/titrax/
application/prs.cww	Windows 下的 CU-Writer	cw、cww	Somchai Prasitjutrakul 博士 somchaip@chulkn.car.chula.ac.th
application/prs.nprend	未知	rnd、rct	John M. Doggett jdoggett@tiac.net
application/remote-printing	包含了远程打印时用于打印机封面的元信息		RFC 1486 Marshall T. Rose mrose@dbc.mtview.ca.us
application/riscos	Acorn 的 RISC OS 二进制文件		RISC OSProgrammer’s Reference Manuals，AcornComputers,Ltd.，ISBN1852501103
application/sdp	SDP 用于描述各种多媒体会话，目的是进行会话声明、会话邀请以及其他形式的多媒体会话初始化		RFC 2327 Henning Schulzrinne hgs@cs.columbia.edu
application/set-payment application/set-payment-initiation application/set-registration application/set-registration-initiation	支持 SET 安全电子交易支付协议		http://www.visa.com http://www.mastercard.com
application/sgml-open-catalog	用于支持 SGML Open TR9401: 1995“实体管理”规范的系统		SGML Open 910 Beaver Grade Road，#3008 Coraopolis，PA 15109 info@sgmlopen.org
application/sieve	Sieve 邮件过滤脚本		RFC 3028

543

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/slate	BBN/Slate 文档格式是作为 BBN/Slate 产品标准文档集的一部分发布的		BBN/Slate Product Mgr BBN Systems and Technologies 10 Moulton Street Cambridge, MA 02138
application/smil	SMIL（同步多媒体集成语言，Synchronized Multimedia Integration Language）将一组独立的多媒体对象整合成了一种同步多媒体表示形式	smi、smil	http://www.w3.org/AudioVideo/
application/tve-trigger	支持在增强型电视接收器里嵌入 URL		"SMPTE: Declarative Data Essence, Content Level 1"，由电影电视工程师协会生成。 http://www.smpte.org
application/vemmi	增强型 videotex 标准		RFC 2122
application/vnd.3M.Post-it-Notes	由 Post-it® 因特网设计者须知因特网控制 / 插件使用	pwn	http://www.3M.com/psnotes/
application/vnd.accpac.simply.aso	Simply Accounting 软件的 7.0 及以上版本。这种类型的文件与 Open Financial Exchange 版本 1.02 的规范一致	aso	http://www.ofx.net
application/vnd.accpac.simply.imp	Simply Accounting 软件的 7.0 及以上版本使用，用于输入它自己的数据	imp	http://www.ofx.net
application/vnd.acucobol	ACUCOBOL-GT Runtime		Dovid Lubin dovid@acucobol.com
application/vnd.aether.imp	支持在 AOL 即时通信、Yahoo! Messenger 或 MSN Messenger 这样的即时消息服务，和运行在无线设备上的特殊即时通信客户端软件集之间进行高效的即时消息通信		有许可证就可以从 Aether 系统中获得 IMP（Wireless Instant Messaging Protocol，无线即时通信协议）规范
application/vnd.anser-Web-certificate-issue-initiation	Web 浏览器装载 ANSER-WEB-Terminal Client 程序的触发器	cii	Hiroyoshi Mori mori@mm.rd.nttdata.co.jp

544

（续）

MIME类型	描　述	扩　展	联系方式与参考文献
application/vnd.anser-Web-funds-transfer-initiation	同上	fti	同上
application/vnd.audiograph	AudioGraph	aep	Horia Cristian H.C.Slusanschi@massey.ac.nz
application/vnd.bmi	CADAM 系统的 BMI 图像格式	bmi	Tadashi Gotoh tgotoh@cadamsystems.co.jp
application/vnd.businessobjects	BusinessObjects4.0 及以上版本	rep	
application/vnd.canon-cpdl application/vnd.canon-lips	支持佳能公司的办公图像产品		Shin Muto shinmuto@pure.cpdc.canon.co.jp
application/vnd.claymore	Claymore.exe	cla	Ray Simpson ray@cnation.com
application/vnd.commerce-battelle	支持一种通用的智能卡卡载信息定界机制，可用于数字商务、身份识别、认证以及基于智能卡的持证人信息交互	ica、icf、icd、icc、ic0、ic1、ic2、ic3、ic4、ic5、ic6、ic7、ic8	David C. Applebaum applebau@131.167.52.15
application/vnd.commonspace	允许通过基于 MIME 的进程正确传输 CommonSpace ™文档。CommonSpace 是由 HoughtonMifflin 公司的 Sixth Floor Media 发布的	csp、cst	Ravinder Chandhok chandhok@within.com
application/vnd.contact.cmsg	用于 CONTACT 软件的 CIM DATABASE	cdbcmsg	Frank Patz fp@contact.de http://www.contact.de
application/vnd.cosmocaller	允许从 Web 站点下载包含了连接参数的文件，调用 CosmoCaller 应用程序对参数进行解释，并初始化与 CosmoCallACD 服务器的连接	cmc	Steve Dellutri sdellutri@cosmocom.com
application/vnd.ctc-posml	Continuum 技术公司的 PosML	pml	Bayard Kohlhepp bayardk@ctcexchange.com

545

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.cups-postscript application/vnd.cups-raster application/vnd.cups-raw	支持 UNIX 通用打印系统的服务器和客户端		http://www.cups.org
application/vnd.cybank	Cybank 数据特有的数据类型		Nor Helmee B. Abd. Halim helmee@cybank.net http://www.cybank.net
application/vnd.dna	DNA 的目标是便捷地将所有 32 位的 Windows 应用程序都转换成 Web 程序	dna	Meredith Searcy msearcy@newmoon.com
application/vnd.dpgraph	由 DpGraph 2000 和 MathWare Cyclone 使用	dpg、mwc dpgraph	David Parker http://www.davidparker.com
application/vnd.dxr	PSI 技术公司的 Digital Xpress Reports	dxr	Michael Duffy miked@psiaustin.com
application/vnd.ecdis-update	支持 ECDIS 应用程序		http://www.sevencs.com
application/vnd.ecowin.chart application/vnd.ecowin.filerequest application/vnd.ecowin.fileupdate application/vnd.ecowin.series application/vnd.ecowin.seriesrequest application/vnd.ecowin.seriesupdate	EcoWin	mag	Thomas Olsson thomas@vinga.se
application/vnd.enliven	支持 Enliven 交互式多媒体的传输	nml	Paul Santinelli psantinelli@narrative.com
application/vnd.epson.esf	精工爱普生 QUASS 流播放器的特有内容	esf	Shoji Hoshina Hoshina.Shoji@exc.epson.co.jp

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.epson.msf	精工爱普生 QUASS 流播放器的特有内容	msf	同上
application/vnd.epson.quickanime	精工爱普生 QuickAnime 播放器的特有内容	qam	Yu Gu guyu@rd.oda.epson.co.jp
application/vnd.epson.salt	精工爱普生 SimpleAnimeLite 播放器的特有内容	slt	Yasuhito Nagatomo naga@rd.oda.epson.co.jp
546 application/vnd.epson.ssf	精工爱普生 QUASS 流播放器的特有内容	ssf	Shoji Hoshina Hoshina.Shoji@exc.epson.co.jp
application/vnd.ericsson.quickcall	电话倍增器快速呼叫	qcall、qca	Paul Tidwell paul.tidwell@ericsson.com http://www.ericsson.com
application/vnd.eudora.data	Eudora 4.3 及以上版本		Pete Resnick presnick@qualcomm.com
application/vnd.fdf	Adobe 表单数据格式		Forms Data Format（表单数据格式），Technical Note 5173，Adobe Systems
application/vnd.ffsns	用于与 FirstFloor 的 Smart Delivery 进行通信的应用程序		Mary Holstege holstege@firstfloor.com
application/vnd.FloGraphIt	NpGraphIt	gph	
application/vnd.framemaker	Adobe FrameMaker 文件	fm、mif、book	http://www.adobe.com
application/vnd.fsc.Weblaunch	支持 Friendly 软件公司的高尔夫模拟软件	fsc	Derek Smith derek@friendlysoftware.com
application/vnd.fujitsu.oasys application/vnd.fujitsu.oasys2	支持富士通的 OASYS 软件	oas	Nobukazu Togashi togashi@ai.cs.fujitsu.co.jp
application/vnd.fujitsu.oasys2	支持富士通的 OASYS V2 软件	oa2	同上
application/vnd.fujitsu.oasys3	支持富士通的 OASYS V5 软件	oa3	Seiji Okudaira okudaira@candy.paso.fujitsu.co.jp
application/vnd.fujitsu.oasysgp	支持富士通的 OASYS GraphPro 软件	fg5	Masahiko Sugimoto sugimoto@sz.sel.fujitsu.co.jp

（续）

MIME类型	描　述	扩　展	联系方式与参考文献
application/vnd.fujitsu.oasysprs	支持富士通的 OASYS 展示软件	bh2	Masumi Ogita ogita@oa.tfl.fujitsu.co.jp
application/vnd.fujixerox.ddd	支持富士施乐的 EDMICS 2000 和 DocuFile	ddd	Masanori Onda Masanori.Onda@fujixerox.co.jp
application/vnd.fujixerox.docuworks	支持富士施乐的 DocuWorks Desk 及 DocuWorks Viewer 软件	xdw	Yasuo Taguchi yasuo.taguchi@fujixerox.co.jp
application/vnd.fujixerox.docuworks.binder	支持富士施乐的 DocuWorks Desk 及 DocuWorks Viewer 软件	xbd	同上
application/vnd.fut-misnet	未知		Jaan Pruulmann jaan@fut.ee
application/vnd.grafeq	让 GrafEq 的用户通过 Web 和电子邮件交换 GrafEq 文档	gqf、gqs	http://www.peda.com
application/vnd.groove-account	Groove 是为小群组交互实现了虚拟空间的对等实体通信系统	gac	Todd Joseph todd_joseph@groove.net
application/vnd.groove-identity-message	同上	gim	同上
application/vnd.groove-injector	同上	grv	同上
application/vnd.groove-tool-message	同上	gtm	同上
application/vnd.groove-tool-template	同上	tpl	同上
application/vnd.groove-vcard	同上	vcg	同上
application/vnd.hhe.lesson-player	支持 LessonPlayer 和 Presentation-Editor 软件	les	Randy Jones Harcourt E-Learning randy_jones@archipelago.com
application/vnd.hp-HPGL	HPGL 文件		*TheHP-GL/2 and HP RTL Referemce Guide*，Addison Wesley，ISBN 0201310147

547

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.hp-hpid	支持惠普的即时传输软件	hpi、hpid	http://www.instant-delivery.com
application/vnd.hp-hps	支持惠普的 WebPrintSmart 软件	hps	http://www.hp.com/go/Webprintsmart_mimetype_specs/
application/vnd.hp-PCL application/vnd.hp-PCLXL	PCL 打印机文件	pcl	"PCL-PJL Technical Reference Manual Documentation Package", HP Part No. 5012-0330
application/vnd.httphone	IP 系统上的 HTTPhone 异步语音		Franck LeFevre franck@k1info.com
application/vnd.hzn-3d-crossword	用于对 Horizon，A Glimpse of Tomorrow 公司的字谜游戏进行加密	x3d	James Minnis james_minnis@glimpse-of-tomorrow.com
application/vnd.ibm.afplinedata	PSF（Print Services Facility，打印服务设施），ACIF（AFP Conversion and Indexing Facility，AFP 转换与索引设施）		Roger Buis buis@us.ibm.com
application/vnd.ibm.MiniPay	MiniPay 认证和支付软件	mpy	Amir Herzberg amirh@vnet.ibm.com
application/vnd.ibm.modcap	Mixed Object Document Content	list3820、listafp、afp、pseg3820	Reinhard Hohensee rhohensee@vnet.ibm.com "MixedObject Document Content Architecture Reference", IBM publication，SC31-6802
application/vnd.informix-visionary	Informix Visionary	vis	Christopher Gales christopher.gales@informix.com
application/vnd.intercon.formnet	支持 Intercon Associates FormNet 软件	xpw、xpx	Thomas A. Gurak assoc@intercon.roc.servtech.com
application/vnd.intertrust.digibox application/vnd.intertrust.nncp	支持用于安全电子商务和数字权限管理的 InterTrust 结构		InterTrust Technologies 460 Oakmead Parkway Sunnyvale, CA 94086 USA info@intertrust.com http://www.intertrust.com

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.intu.qbo	仅用于 QuickBooks 6.0（加拿大）	qbo	Greg Scratchley greg_scratchley@intuit.com Open Financial Exchange 规范中讨论的文件的格式，可从 http://www.ofx.net 上获得
application/vnd.intu.qfx	仅用于 Quicken 99 及后继版本	qfx	同上
application/vnd.is-xpr	搜信公司的 Express	xpr	Satish Natarajan satish@infoseek.com
application/vnd.japannet-directory-service application/vnd.japannet-jpnstore-wakeup application/vnd.japannet-payment-wakeup application/vnd.japannet-registration application/vnd.japannet-registration-wakeup application/vnd.japannet-setstore-wakeup application/vnd.japannet-verification application/vnd.japannet-verification-wakeup	支持 三菱电机公司的 JapanNet 安全、认证和支付软件		Jun Yoshitake yositake@iss.isl.melco.co.jp
application/vnd.koan	在 SSEYO Koan Netscape Plugin 这样的辅助程序的帮助下，在因特网上自动重放 Koan 音乐文件	skp、skd skm、skt	Peter Cole pcole@sseyod.demon.co.uk
application/vnd.lotus-1-2-3	Lotus-1-2- 和 Lotus Approach	123、wk1 wk3、wk4	Paul Wattenberger Paul_Wattenberger@lotus.com

548

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.lotus-approach	Lotus Approach	apr、vew	同上
application/vnd.lotus-freelance	Lotus Freelance	prz、pre	同上
application/vnd.lotus-notes	Lotus Notes	nsf、ntf、ndl、ns4 ns3、ns2 nsh、nsg	Michael Laramie laramiem@btv.ibm.com
application/vnd.lotus-organizer	Lotus Organizer	or3、or2、org	Paul Wattenberger Paul_Wattenberger@lotus.com
application/vnd.lotus-screencam	Lotus ScreenCam	scm	同上
application/vnd.lotus-wordpro	Lotus WordPro	lwp、sam	同上
549 application/vnd.mcd	Micro CADAM 公司的 CAD 软件	mcd	Tadashi Gotoh tgotoh@cadamsystems.co.jp http://www.cadamsystems.co.jp
application/vnd.mediastation.cdkey	支持 Media Station 公司的 CDKey 远程 CDROM 通信协议	cdkey	Henry Flurry henryf@mediastation.com
application/vnd.meridian-slingshot	Meridian 数据公司的 Slingshot		Eric Wedel Meridian Data, Inc. 5615 Scotts Valley Drive Scotts Valley, CA 95066 ewedel@meridian-data.com
application/vnd.mif	FrameMaker 交换格式	mif	ftp://ftp.frame.com/pub/techsup/techinfo/dos/mif4.zip Mike Wexler Adobe Systems, Inc333 W. San Carlos St. San Jose, CA 95110 USA mwexler@adobe.com
application/vnd.minisoft-hp3000-save	NetMail 3000 保存格式		Minisoft，Inc support@minisoft.com ftp://ftp.3k.com/DOC/ms92-save-format.txt

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/ vnd.mitsubishi. misty-guard. trustWeb	支持三菱电机公司的 TrustWeb 软件		Manabu Tanaka mtana@iss.isl.melco.co.jp
application/vnd. Mobius.DAF	支持 Mobius Managenect System 软件	daf	Celso Rodriguez crodrigu@mobius.com Greg Chrzczon gchrzczo@mobius.com
application/vnd. Mobius.DIS	同上	dis	同上
application/vnd. Mobius.MBK	同上	mbk	同上
application/vnd. Mobius.MQY	同上	mqy	同上
application/vnd. Mobius.MSL	同上	msl	同上
application/vnd. Mobius.PLC	同上	plc	同上
application/vnd. Mobius.TXF	同上	txf	同上
application/ vnd.motorola. flexsuite	FLEXsuite ™是无线报文协议的集合。无线报文服务提供商的网关、无线 OS 和应用程序都使用此类型		Mark Patton Motorola Personal Networks Group fmp014@email.mot.com 在适当许可协议的支持下，可以从摩托罗拉获得 FLEXsuite ™规范
application/ vnd.motorola. flexsuite.adsi	FLEXsuite ™是无线报文协议的集合。这种类型为各种数据加密解决方案提供了一种无线友好格式		同上
application/ vnd.motorola. flexsuite.fis	FLEXsuite ™是无线报文协议的集合。这种类型是向无线设备高效传输结构化信息（比如新闻、股票、天气）的无线友好格式		同上
application/ vnd.motorola. flexsuite.gotap	FLEXsuite ™是无线报文协议的集合 这种类型为通过空中报文进行无线设备属性编程提供了一种通用的无线友好格式		同上

550

（续）

MIME类型	描述	扩展	联系方式与参考文献
application/vnd.motorola.flexsuite.kmr	FLEXsuite ™是无线报文协议的集合。这种类型为加密密钥管理提供了一种无线友好格式		同上
application/vnd.motorola.flexsuite.ttc	FLEXsuite ™是无线报文协议的集合。这种类型支持在通过标记文本压缩进行高效文本传输时使用的一种无线友好格式		同上
application/vnd.motorola.flexsuite.wem	FLEXsuite ™是无线报文协议的集合。这种类型为向无线设备传输因特网电子邮件提供了无线友好格式		同上
application/vnd.mozilla.xul+xml	支持 Mozilla 的因特网应用程序集	xul	Dan Rosen2 dr@netscape.com
application/vnd.ms-artgalry	支持微软的 Art Gallery	cil	deansl@microsoft.com
application/vnd.ms-asf	ASF 是一种多媒体文件格式，要通过网络以流的形式传输其内容以支持分布式多媒体应用程序。ASF 内容中可包含任意媒体类型的任意组合（比如音频、视频、图片、URL、HTML 内容、MIDI、二维和三维模式、脚本以及各种类型的对象）	asf	Eric Fleischman ericf@microsoft.com http://www.microsoft.com/mind/0997/netshow/netshow.asp
application/vnd.ms-excel	微软的 Excel 电子表格	xls	Sukvinder S. Gill sukvg@microsoft.com
application/vnd.ms-lrm	微软的特有格式	lrm	Eric Ledoux ericle@microsoft.com
application/vnd.ms-powerpoint	微软的 PowerPoint 演示文稿	ppt	Sukvinder S. Gill sukvg@microsoft.com
551 application/vnd.ms-project	微软的项目文件	mpp	同上
application/vnd.ms-tnef	标识通常只能由 MAPI 应用程序处理的附件。这种类型是 Rich Text 本和 lcon 信息等富 MAPI 属性的封装格式，如果不把这些信息封装起来，就会在报文传输中降级		同上

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.ms-works	微软的 Works 软件		同上
application/vnd.mseq	MSEQ 是一种适用于无线设备的压缩多媒体格式	mseq	Gwenael Le Bodic Gwenael.le_bodic@alcatel.fr http://www.3gpp.org
application/vnd.msign	由那些实现了 msign 协议的应用程序使用，要求移动设备提供签名		Malte Borcherding Malte.Borcherding@brokat.com
application/vnd.music-niff	NIFF 音乐文件		Cindy Grande 72723.1272@compuserve.com ftp://blackbox.cartah.washington.edu/pub/NIFF/NIFF6A.TXT
application/vnd.musician	RenaiScience 公司设计并开发的 MUSICIAN 乐谱语言 / 编码系统	mus	Robert G. Adams gadams@renaiscience.com
application/vnd.netfpx	用于多分辨率图片信息的动态检索，就像惠普公司的 Imaging for Internet 软件一样	fpx	Andy Mutz andy_mutz@hp.com
application/vnd.noblenet-directory	支持 RogueWave 购买的 NobleNet Directory 软件	nnd	http://www.noblenet.com
application/vnd.noblenet-sealer	支持 RogueWave 购买的 NobleNet Sealer 软件	nns	http://www.noblenet.com
application/vnd.noblenet-Web	支持 RogueWave 购买的 NobleNet Web 软件	nnw	http://www.noblenet.com
application/vnd.novadigm.EDM	支持 Novadigm 的 RADIA 和 EDM 产品	edm	Phil Burgard pburgard@novadigm.com
application/vnd.novadigm.EDX	同上	edx	同上
application/vnd.novadigm.EXT	同上	ext	同上
application/vnd.osa.netdeploy	支持 Open Software Associates 的 netDeploy 应用程序部署软件	ndc	Steve Klos stevek@osa.com http://www.osa.com
application/vnd.palm	PalmOS 系统软件和应用程序使用——这种新的类型 application/vnd.palm，取代了老的类型 application/x-pilot	prc、pdb、pqa、oprc	Gavin Peacock gpeacock@palm.com

552

（续）

MIME类型	描　述	扩　展	联系方式与参考文献
application/vnd.pg.format	宝洁公司特有的标准报告系统	str	April Gandert TN152 Procter & Gamble Way Cincinnati, Ohio 45202 (513) 983-4249
application/vnd.pg.osasli	宝洁公司特有的标准报告系统	ei6	同上
application/vnd.powerbuilder6 application/vnd.powerbuilder6-s application/vnd.powerbuilder7 application/vnd.powerbuilder7-s application/vnd.powerbuilder75 application/vnd.powerbuilder75-s	仅在赛贝斯公司的 PowerBuilder 发行版本 6、7 和 7.5 运行时环境中使用，包括不安全和安全两种版本	pbd	Reed Shilts reed.shilts@sybase.com
application/vnd.previewsystems.box	Preview Systems 公司的 ZipLock/VBox 产品	box、vbox	Roman Smolgovsky romans@previewsystems.com http://www.previewsystems.com
application/vnd.publishare-delta-tree	在 Capella 计算机的 PubliShare 运行时环境中使用	qps	Oren Ben-Kiki publishare-delta-tree@capella.co.il
application/vnd.rapid	Emultek 的快速封装应用程序	zrp	Itay Szekely etay@emultek.co.il
application/vnd.s3sms	将 Sonera SmartTrust 产品的传输机制整合进因特网架构		Lauri Tarkkala Lauri.Tarkkala@sonera.com http://www.smarttrust.com
application/vnd.seemail	支持对 SeeMail 文件的传输。SeeMail 是一种应用程序，它可捕获视频和音频，并通过按位压缩方式将这两个部分压缩存档到一个文件	see	Steven Webb steve@wynde.com http://www.realmediainc.com
application/vnd.shana.informed.formdata	Shana 电子表格数据格式	ifm	Guy Selzler Shana Corporation gselzler@shana.com
application/vnd.shana.informed.formtemp	Shana 电子表格数据格式	itp	同上

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.shana.informed.interchange	Shana 电子表格数据格式	iif、iifl	同上
application/vnd.shana.informed.package	Shana 电子表格数据格式	ipk、ipkg	同上
application/vnd.street-stream	Street 技术公司特有的格式		Glenn Levitt Street Technologies streetd1@ix.netcom.com 553
application/vnd.svd	Dateware 电子公司的 SVD 文件		Scott Becker dataware@compumedia.com
application/vnd.swiftview-ics	支持 SwiftView®		Randy Prakken tech@ndg.com http://www.ndg.com/svm.htm
application/vnd.triscape.mxs	支持 Triscape Map 浏览器	mxs	Steven Simonoff scs@triscape.com
application/vnd.trueapp	True BASIC 文件	tra	J. Scott Hepler scott@truebasic.com
application/vnd.truedoc	Bitstream 公司特有的格式		Brad Chase brad_chase@bitstream.com
application/vnd.ufdl	UWI 的 UFDL 文件	ufdl、ufd、frm	Dave Manning dmanning@uwi.com http://www.uwi.com/
application/vnd.uplanet.alert application/vnd.uplanet.alert-wbxml application/vnd.uplanet.bearer-choi-wbxml application/vnd.uplanet.bearer-choice application/vnd.uplanet.cacheop application/vnd.uplanet.cacheop-wbxml	Unwired Planet（现在的 Openwave）公司为移动设备提供的微浏览器 UP 所使用的格式		iana-registrar@uplanet.com http://www.openwave.com

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.uplanet.channel application/vnd.uplanet.channel-wbxml application/vnd.uplanet.list application/vnd.uplanet.list-wbxml application/vnd.uplanet.listcmd application/vnd.uplanet.listcmd-wbxml application/vnd.uplanet.signal			
application/vnd.vcx	VirtualCatalog	vcx	Taisuke Sugimoto sugimototi@noanet.nttdata.co.jp
application/vnd.vectorworks	VectorWorks 的图像文件	mcd	Paul C. Pharr pharr@diehlgraphsoft.com
application/vnd.vidsoft.vidconference	VidConference 格式	vsc	Robert Hess hess@vidsoft.de
554 application/vnd.visio	Visio 文件	vsd、vst、vsw、vss	Troy Sandal troys@visio.com
application/vnd.vividence.scriptfile	Vividence 文件	vsf、vtd、vd	Mark Risher markr@vividence.com
application/vnd.wap.sic	WAP Service Indication 格式	sic、wbxml	WAP Forum Ltd http://www.wapforum.org
application/vnd.wap.slc	WAP Service Loading 格式。与 Service Loading 规范相符的所有内容都可以在 http://www.wapforum.org 上找到	slc、wbxml	同上
application/vnd.wap.wbxml	无线设备的 WAP WBXML 二进制 XML 格式	wbxml	同上 “WAP BinaryXMLContent Format——WBXML vevsion1.1”

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/vnd.wap.wmlc	无线设备的 WAP WML 格式	wmlc、wbxml	同上
application/vnd.wap.wmlscriptc	WAP WMLScript 格式	wmlsc	同上
application/vnd.Webturbo	WebTurbo 格式	wtb	Yaser Rehem Sapient Corporation yrehem@sapient.com
application/vnd.wrq-hp3000-labelled	支持 HP3000 格式		support@wrq.com support@3k.com
application/vnd.wt.stf	支持 Worldtalk 软件	stf	Bill Wohler wohler@worldtalk.com
application/vnd.xara	CorelXARA 保存的 Xara 文件，这是 Xara 有限公司编写（并由 Corel 销售的）的面向对象的矢量图像包	xar	David Matthewman david@xara.com http://www.xara.com
application/vnd.xfdl	UWI 的 XFDL 文件	xfdl、xfd、frm	Dave Manning dmanning@uwi.com http://www.uwi.com
application/vnd.yellowriver-custom-menu	支持 Yellow River CustomMenu 插件，这个插件提供了自定义的浏览器下拉菜单	cmp	yellowriversw@yahoo.com
application/whoispp-query	在 MIME 中定义了 Whois++ 协议的查询方式		RFC 2957
application/whoispp-response	在 MIME 中定义了 Whois++ 协议的响应方式		RFC 2958
application/wita	Wang（王安）信息传输结构		文档号 715-0050A，Wang Laboratories（王安实验室） campbell@redsox.bsw.com
application/wordperfect5.1	WordPerfect 文档		
application/x400-bp	承载了所有没有注册 IANA 映射的 X.400 主体部分		RFC 1494
application/x-bcpio	老版的二进制 CPIO 归档格式	bcpio	

555

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
application/ x-cdlink	允许在 Web 页面内整合 CD-ROM 媒体	vcd	http://www.cdlink.com
application/ x-chess-pgm	来自 Apache mime.types	pgn	
application/ x-compress	Unix 压缩的二进制数据	z	
application/ x-cpio	CPIO 归档文件	cpio	
application/ x-csh	CSH 脚本	csh	
application/ x-director	Macromedia 的 Director 档案	dcr、dir、dxr	
application/ x-dvi	TeX DVI 文件	dvi	
application/ x-futuresplash	来自 Apache 的 mime.types	spl	
application/ x-gtar	GNU 的 tar 归档文件	gtar	
application/ x-gzip	GZIP 压缩数据	gz	
application/ x-hdf	来自 Apache 的 mime.types	hdf	
application/ x-javascript	JavaScript 文件	js	
application/ x-koan	在 SSEYO Koan Netscape Plugin 辅助应用程序的帮助下，支持在因特网上自动重放 Koan 音乐文件	skp、skd skt、skm	
application/ x-latex	LaTeX 文件	latex	
application/ x-netcdf	NETCDF 文件	nc、cdf	
application/x-sh	SH 脚本	sh	
application/ x-shar	SHAR 归档文件	shar	

（续）

MIME类型	描　述	扩　展	联系方式与参考文献
application/x-shockware-flash	Macromedia 的 Flash 文件	swf	
application/x-stuffit	Stufflt 归档文件	sit	
application/x-sv4cpio	Unix SysV R4 CPIO 归档文件	sv4cpio	
application/x-sv4crc	Unix SysV R4 CPIO 带 CRC 校验的归档文件	sv4crc	
application/x-tar	TAR 归档文件	tar	
application/x-tcl	TCL 脚本	tcl	
application/x-tex	TeX 文件	tex	
application/x-texinfo	TeX 信息文件	texinfo、texi	
application/x-troff	TROFF 文件	t、tr、roff	
application/x-troff-man	TROFF Unix 帮助手册	man	
application/x-troff-me	TROFF+me 文件	me	
application/x-troff-ms	TROFF+ms 文件	ms	
application/x-ustar	扩展 tar 交互格式	ustar	参见 IEEE 1003.1(1990) 规范
application/x-wais-source	WAIS 源文件结构	Src	
application/xml	可扩展标记语言格式文件（如果希望浏览器等浏览工具将文件作为纯文本处理，就使用 text/xml）	xml、dtd	RFC 2376
application/zip	PKWARE zip 归档文件	zip	

556

D.4.2 audio/*

表 D-4 总结了音频内容类型。

表D-4 “Audio”MIME类型

MIME类型	描　　述	扩　　展	联系方式与参考文献
audio/32kadpcm	32 kHz ADPCM 音频编码		RFC 2421
audio/basic	8kHz 单声道 8 位 ISDN u-lawPCM 编码的音频	au、snd	RFC 1341
audio/G.772.1	G.722.1 将 50Hz-7kHz 的音频信号压缩为 24kbit/s 或 32kbit/s。可用于语音、音乐和其他类型的音频		RFC 3047
audio/L16	Audio/L16 基于 RFC 1890 中描述的 L16。L16 表示以 16 位带符号表示法表示的非压缩音频数据		RFC 2586
audio/MP4A-LATM	MPEG-4 音频		RFC 3016
audio/midi	MIDI 音乐文件	mid、midi、kar	
audio/mpeg	MPEG 编码音频文件	mpga、mp2、mp3	RFC 3003
audio/parityfec	RTP 音频的奇偶前向纠错码		RFC 3009
audio/prs.sid	科摩多公司的 64 SID 音频文件	sid、psid	http://www.geocities.com/SiliconValley/Lakes/5147/sidplay/docs.html#fileformats
audio/telephoneevent	逻辑电话事件		RFC 2833
audio/tone	电话声音模式		RFC 2833
audio/vnd.cns.anp1	支持 Comverse 网络系统公司 Access NP 网络服务平台上的声音和统一报文应用程序特性		Ann McLaughlin Comverse 网络系统 amclaughlin@comversens.com
audio/vnd.cns.inf1	支持 Comverse 网络系统公司 TRILOGUE Infinity 网络服务平台上的声音和统一报文应用程序特性		同上

557

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
audio/vnd.digitalwinds	Digital Winds 音乐是封装在很小的封包（<3K）内的永不终结、可再生的交互式 MIDI 音乐	eol	Armands Strazds armands.strazds@medienhaus-bremen.de
audio/vnd.everad.plj	EverAD 音频编码的特有格式	plj	Tomer Weisberg tomer@everad.com
audio/vnd.lucent.voice	包含朗讯科技的 Intuity ™ AUDIX® Multimedia Messaging System 和 Lucent Voice Player 在内的声音报文	lvp	Frederick Block rickblock@lucent.com http://www.lucent.com/lvp/
audio/vnd.nortel.vbk	北电网络公司的声音块音频编码特有的格式	vbk	Glenn Parsons Glenn.Parsons@NortelNetworks.com
audio/vnd.nuera.ecelp4800	纽亚通信的音频和语音编码所特有的格式，用于纽亚的 VoIP 网关、终端、应用程序服务器，还作为各种主机平台和 OS 的媒体服务使用	ecelp4800	Michael Fox mfox@nuera.com
audio/vnd.nuera.ecelp7470	同上	ecelp7470	同上
audio/vnd.nuera.ecelp9600	同上	ecelp9600	同上
audio/vnd.octel.sbc	朗讯科技的 Sierra ™、Overture ™和 IMA ™ 平台上的声音报文使用的平均值为 18kbps 的可变速率编码		Jeff Bouis jbouis@lucent.com
audio/vnd.qcelp	高通的音频编码	qcp	Andy Dejaco adejaco@qualcomm.com
audio/vnd.rhetorex.32kadpcm	在朗讯科技的 CallPerformer ™、Unified Messenger ™这样的声音报文产品和其他产品中使用的 32kbps Rhetorex ™ ADPCM 音频编码		Jeff Bouis jbouis@lucent.com
audio/vnd.vmx.cvsd	在包含朗讯科技的 Overture200 ™、Overture 300 ™和 VMX 300 ™产品线在内的声音报文产品中使用的音频编码		同上

（续）

MIME类型	描　　述	扩　　展	联系方式与参考文献
audio/x-aiff	AIFF 音频文件格式	aif, aiff, aifc	
audio/x-pn-realaudio	Real Networks（前身是 Progressive Networks）公司的 RealAudio 元文件格式	ram, rm	
558 audio/x-pn-realaudio-plugin	来自 Apache 的 mime.types	rpm	
audio/x-realaudio	Real Networks（前身是 Progressive Networks）公司的 RealAudio 音频格式	ra	
audio/x-wav	WAV 音频文件	wav	

D.4.3　chemical/*

表 D-5 中很多信息都是从 Chemical MIME 主页上获得的（http://www.ch.ic.ac.uk/chemime/）。

表D-5　“Chemical”MIME类型

MIME类型	描　　述	扩展	联系方式和参考文献
chemical/x-alchemy	Alchemy 格式	alc	http://www.camsoft.com
chemical/x-cache-csf		csf	
chemical/x-cactvs-binary	CACTVS 二进制格式	cbin	http://cactvs.cit.nih.gov
chemical/x-cactvs-ascii	CACTVS ASCII 格式	cascii	http://cactvs.cit.nih.gov
chemical/x-cactvs-table	CACTVS 表格式	ctab	http://cactvs.cit.nih.gov
chemical/x-cdx	ChemDraw eXchange 文件	cdx	http://www.camsoft.com
chemical/x-cerius	MSI Cerius II 格式	cer	http://www.msi.com
chemical/x-chemdraw	ChemDraw 文件	chm	http://www.camsoft.com
chemical/x-cif	Crystallographic Interchange Format	cif	http://www.bernstein-plus-sons.com/software/rasmol/ http://ndbserver.rutgers.edu/NDB/mmcif/examples/index.html
chemical/x-mmcif	高分子 CIF	mcif	同上
chemical/x-chem3d	Chem3D 格式	c3d	http://www.camsoft.com
chemical/x-cmdf	CrystalMaker 数据格式	cmdf	http://www.crystalmaker.co.uk

（续）

MIME类型	描　　述	扩展	联系方式和参考文献
chemical/x-compass	Takahashi 的 Compass 程序	cpa	
chemical/x-crossfire	Crossfire 文件	bsd	
chemical/x-cml	化学标记语言	cml	http://www.xml-cml.org
chemical/x-csml	化学风格标记语言	csml、csm	http://www.mdli.com
chemical/x-ctx	Gasteiger 小组的 CTX 文件格式	ctx	
chemical/x-cxf		cxf	
chemical/x-daylight-smiles	Smiles 格式	smi	http://www.daylight.com/dayhtml/smiles/index.html
chemical/x-embl-dl-nucleotide	EMBL 核苷酸格式	emb	http://mercury.ebi.ac.uk
chemical/x-galactic-spc	光谱和色谱数据的 SPC 格式	spc	http://www.galactic.com/galactic/Data/spcvue.htm
chemical/x-gamess-input	GAMESS 输入格式	inp、gam	http://www.msg.ameslab.gov/GAMESS/Graphics/MacMolPlt.shtml
chemical/x-gaussian-input	Gaussian Input 格式	gau	http://www.mdli.com
chemical/x-gaussian-checkpoint	Gaussian Checkpoint 格式	fch、fchk	http://products.camsoft.com
chemical/x-gaussian-cube	Gaussian Cube（Wavefunction 波函数）格式	cub	http://www.mdli.com
chemical/x-gcg8-sequence		gcg	
chemical/x-genbank	ToGenBank 格式	gen	
chemical/x-isostar	分子间相互作用的 IsoStar Library	istr、ist	http://www.ccdc.cam.ac.uk
chemical/x-jcamp-dx	JCAMP Spectroscopic Data Exchange 格式	jdx、dx	http://www.mdli.com
chemical/x-jjc-review-surface	Re_View3 Orbital Contour 文件	rv3	http://www.brunel.ac.uk/depts/chem/ch241s/re_view/rv3.htm
chemical/x-jjc-review-xyz	Re_View3Animation 文件	xyb	http://www.brunel.ac.uk/depts/chem/ch241s/re_view/rv3.htm
chemical/x-jjc-review-vib	Re_View3 Vibration（振动）文件	rv2、vib	http://www.brunel.ac.uk/depts/chem/ch241s/re_view/rv3.htm

559

（续）

MIME类型	描　　述	扩展	联系方式和参考文献
chemical/x-kinemage	Kinetic（Protein Structure，蛋白质结构）图片	kin	http://www.faseb.org/protein/kinemages/MageSoftware.html
chemical/x-macmolecule	高分子文件格式	mcm	
chemical/x-macromodel-input	宏模型分子力学	mmd、mmod	http://www.columbia.edu/cu/chemistry/
chemical/x-mdl-molfile	MDL Molfile	mol	http://www.mdli.com
chemical/x-mdl-rdfile	Reaction（反应）数据文件	rd	http://www.mdli.com
chemical/x-mdl-rxnfile	MDL Reaction 格式	rxn	http://www.mdli.com
chemical/x-mdl-sdfile	MDL Structure 数据文件	sd	http://www.mdli.com
chemical/x-mdl-tgf	MDL Transportable Graphics 格式	tgf	http://www.mdli.com
chemical/x-mif		mif	
chemical/x-mol2	SYBYL 分子的便携式表示形式	mol2	http://www.tripos.com
chemical/x-molconn-Z	Molconn-Z 格式	b	http://www.eslc.vabiotech.com/molconn/molconnz.html
chemical/x-mopac-input	MOPAC Input 格式	mop	http://www.mdli.com
chemical/x-mopac-graph	MOPAC Graph 格式	gpt	http://products.camsoft.com
chemical/x-ncbi-asn1		asn（老格式）	
chemical/x-ncbi-asn1-binary		val	
560 chemical/x-pdb	蛋白质数据库 pdb	pdb	http://www.mdli.com
chemical/x-swissprot	SWISS-PROT 蛋白质序列数据库	sw	http://www.expasy.ch/spdbv/text/download.htm
chemical/x-vamas-iso14976	材料及标准的凡尔赛协定	vms	http://www.acolyte.co.uk/JISO/
chemical/x-vmd	视觉分子动力学软件	vmd	http://www.ks.uiuc.edu/Research/vmd/
chemical/x-xtel	Xtelplot 文件格式	xtel	http://www.recipnet.indiana.edu/graphics/xtelplot/xtelplot.htm
chemical/x-xyz	Co-ordinate Animation 格式	xyz	http://www.mdli.com

D.4.4 image/*

表 D-6 总结了电子邮件和 HTTP 经常交换的一些图片类型。

表D-6 “Image”MIME类型

MIME类型	描　述	扩　展	联系方式和参考文献
image/bmp	Windows 的 BMP 图片格式	bmp	
image/cgm	CGM（Computer Graphics Metafile，计算机图形元文件）是便携式存储和传输二维插图的国际标准		Alan Francis A.H.Francis@open.ac.uk 参见 ISO 8632:1992，IS 8632:1992 Amendment1（1994）和 IS 8632:1992 Amendment2（1995）
image/g3fax	G3 传真字节流		RFC 1494
image/gif	Compuserve GIF 图片	gif	RFC 1341
image/ief		ief	RFC 1314
image/jpeg	JPEG 图片	jpeg、jpg、jpe、jfif	JPEG 草案标准 ISO 10918-1 CD
image/naplps	NAPLPS（北美表示层协议语法，North American Presentation Layer Protocol Syntax）图片		ANSI X3.110-1983 CSA T500-1983
image/png	PNG（便携式网络图像）图片	png	因特网草案 draft-boutell-png-spec-04.txt，“Png Specification Version1.0”
image/prs.btif	美国国家银行为支票及其他应用程序的 BTIF 图片视图所使用的格式	btif、btf	Arthur Rubin arthurr@crt.com
image/prs.pti	PTI 编码图片	pti	Juern Laun juern.laun@gmx.de http://server.hvzgymn.wn.sch-ule-bw.de/pti/
image/tiff	TIFF 图片	tiff、tif	RFC 2302 561
image/vnd.cns.inf2	支持 Comverse 网络系统公司 TRILOGUE Infinity 网络服务器平台上可用的应用程序特性		Ann McLaughlin Comverse 网络系统公司 amclaughlin@comversens.com
image/vnd.dxf	DXF 矢量 CAD 文件	dxf	

（续）

MIME类型	描　　述	扩　　展	联系方式和参考文献
image/vnd.fastbidsheet	FastBid Sheet 中包含一个用来表示工程或建筑图纸的光栅或矢量图片	fbs	Scott Becker scottb@bxwa.com
image/vnd.fpx	柯达的 FlashPix 图片	fpx	Chris Wing format_change_request@kodak.com http://www.kodak.com
image/vnd.fst	FAST 搜索及传输的图片格式	fst	Arild Fuldseth Arild.Fuldseth@fast.no
image/vnd.fujixerox.edmics-mmr	富士施乐的 EDMICS MMR 图片格式	mmr	Masanori Onda Masanori.Onda@fujixerox.co.jp
image/vnd.fujixerox.edmics-rlc	富士施乐的 EDMICS RLC 图片格式	rlc	同上
image/vnd.mix	MIX 文件包含了流中用来表示图片及相关信息的二进制数据。由微软的 PhotDraw 和 PictureIt 软件使用		Saveen Reddy2 saveenr@microsoft.com
image/vnd.net-fpx	柯达的 FlashPix 图片		Chris Wing format_change_request@kodak.com http://www.kodak.com
image/vnd.wap.wbmp	来自 Apache 的 mime.types	wbmp	
image/vnd.xiff	Pagis 软件所用的扩展图片格式	xif	Steve Martin smartin@xis.xerox.com
image/x-cmu-raster	来自 Apache 的 mime.types	ras	
image/x-portable-anymap	PBM 通用图片	pnm	Jeff Poskanzer http://www.acme.com/software/pbmplus/
image/x-portable-bitmap	PBM 位图图片	pbm	同上
image/x-portable-graymap	PBM 灰度图片	pgm	同上
image/x-portable-pixmap	PBM 彩色图片	ppm	同上
image/x-rgb	硅图的 RGB 图片	rgb	
image/x-xbitmap	X-Window System 的位图图片	xbm	
image/x-xpixmap	X-Window System 的彩色图片	xpm	
562 image/x-xwindowdump	X-Window System 的屏幕截取图片	xwd	

D.4.5　message/*

报文是用来（通过电子邮件、HTTP 或其他传输协议）交互数据对象的复合类型。表 D-7 描述了常见的 MIME 报文类型。

表D-7　“Message”MIME类型

MIME类型	描　　述	扩　　展	联系方式及参考文献
message/delivery-status			
message/disposition-notification			RFC 2298
message/external-body			RFC 1341
message/http			RFC 2616
message/news	定义了一种通过电子邮件传输可读新闻文章的方式——新闻首部的语义超出了 RFC 822 定义的范围，所以仅使用 message/rfc822 是不够的		RFC 1036
message/partial	允许对那些太大，无法直接通过电子邮件传输的主体部分进行分段传输		RFC 1341
message/rfc822	完整的电子邮件报文		RFC 1341
message/s-http	安全 HTTP 报文，是对 SSL 上 HTTP 的替代方式		RFC 2660

D.4.6　model/*

MIME 类型 model 是 IETF 注册的扩展类型。它表示的是物理世界的数学模型，用于计算机辅助设计和三维图像。表 D-8 描述了部分 model 格式。

表D-8　“Model”MIME类型

MIME类型	描　　述	扩　　展	联系方式及参考文献
model/iges	IGES（Initial Graphics Exchange Specification，初始图像交换规范）定义了一种中立数据格式，通过它可以在 CAD 系统之间进行信息的数字化交换	igs、iges	RFC 2077
model/mesh		msh、mesh、silo	RFC 2077

（续）

MIME类型	描　　述	扩　　展	联系方式及参考文献
model/vnd.dwf	DWF CAD 文件	dwf	Jason Pratt jason.pratt@autodesk.com
model/vnd.flatland.3dml	支持 Flatland 产品的 3DML 模型	3dml、3dm	Michael Powers pow@flatland.com http://www.flatland.com
model/vnd.gdl model/vnd.gs-gdl	GDL（Geometric Description Language，几何描述语言）是 Graphisoft 为 ArchiCAD 设计的参数化对象定义语言	gdl、gsm、win、dor、lmp、rsm、msm、ism	Attila Babits ababits@graphisoft.hu http://www.graphisoft.com
model/vnd.gtw	Gen-Trix 模型	gtw	Yutaka Ozaki yutaka_ozaki@gen.co.jp
model/vnd.mts	Virtue 的 MTS 模型格式	mts	Boris Rabinovitch boris@virtue3d.com
model/vnd.parasolid.transmit.binary	二进制 Parasolid 建模文件	x_b	http://www.ugsolutions.com/products/parasolid/
model/vnd.parasolid.transmit.text	文本 Parasolid 建模文件	x_t	http://www.ugsolutions.com/products/parasolid/
model/vnd.vtu	Virtue 的 VTU 模型格式	vtu	Boris Rabinovitch boris@virtue3d.com
model/vrml	虚拟现实标记语言格式的文件	wrl、vrml	RFC 2077

563

D.4.7　multipart/*

MIME 类型 multipart 是包含有其他对象的组合对象。子类型描述了多部分封装的实现，以及如何处理组件。表 D-9 总结了 Multipart 媒体类型。

表D-9　“Multipart”MIME类型

MIME类型	描　　述	扩　　展	联系方式及参考文献
multipart/alternative	内容包括一个可替换表达方式列表，每种方式都有自己的 `Content-Type`。客户端可以选择其支持得最好的组件		RFC 1341
multipart/appledouble	苹果 Macintosh 文件包含了“资源分支”和其他用来描述实际文件内容的桌面数据。这个由多部分组成的内容在一个部分中发送 Apple 元数据，并在另一个部分中发送实际内容		http://www.isi.edu/in-notes/iana/assignments/media-types/multipart/appledouble

（续）

MIME类型	描　　述	扩　　展	联系方式及参考文献
multipart/byteranges	HTTP 报文中包含了多个范围的内容时，要放在 multipart/byteranges 对象中传输。这个媒体类型包含由 MIME 边界分隔的两个或多个部分，每个部分都有自己的 Content-Type 和 Content-Range 字段		RFC 2068
multipart/digest	包含一组以易读形式表示的个人电子邮件报文		RFC 1341
multipart/encrypted	用两个部分来支持密码加密的内容。第一部分包含了解密第二主体部分的数据所必须的控制信息，它是根据协议参数值标记的。第二部分包含了 application/octet-stream 类型的加密数据		RFC 1847
multipart/form-data	根据用户填表的结果将一组值封装起来		RFC 2388
multipart/header-set	从任意描述性元数据中将用户数据分离出来		http://www.isi.edu/in-notes/iana/assignments/media-types/multipart/header-set
multipart/mixed	一组对象		RFC 1341
multipart/parallel	语法与 multipart/mixed 相同，但在能够使用它的系统中，要同时提供所有的部分		RFC 1341
multipart/related	供包含了几个相互关联的主体部分的复合对象使用。主体各部分之间的关系将其与其他对象类型区分开来。这些关系通常是由引用其他组件的对象组件内部链接表示的		RFC 2387
multipart/report	为各种类型的电子邮件报告定义了一种通用的容器类型		RFC 1892
multipart/signed	使用两个部分来支持经过密码签名的内容。第一部分是包含了其 MIME 首部的内容；第二部分包含了验证数字签名所需的信息		RFC 1847
multipart/voice-message	提供了一种机制，可将声音报文封装到一个标记为 VPIM v2 兼容的容器中		RFC 2421 和 RFC 2423

564

D.4.8 text/*

565 Text 媒体类型包含了字符和潜在的格式化信息。表 D-10 总结了 Text MIME 类型。

表D-10 “Text”MIME类型

MIME类型	描 述	扩 展	联系方式和参考文献
text/calendar	支持 iCalendar 日历和日程标准		RFC 2445
text/css	层级样式表	css	RFC 2318
text/directory	装载 LDAP 这样的目录数据库中的记录数据		RFC 2425
text/enriched	简单的格式化文本，支持字体、颜色和间距方式。使用类 SGML 的标记来开始或结束格式化工作		RFC 1896
text/html	HTML 文件	html、htm	RFC 2854
text/parityfec	RTP 流中文本流的前向纠错码		RFC 3009
text/plain	普通的文本	asc、txt	
text/prs.lines.tag	支持用于电子邮件注册的标记表格	tag、dsc	John Lines john@paladin.demon.co.uk http://www.paladin.demon.co.uk/tag-types/
text/rfc822-headers	就像邮件发送失败报告那样，用来绑定一组电子邮件首部		RFC 1892
text/richtext	富文本的较老形式。参见 text/enriched	rtx	RFC 1341
text/rtf	RTF（Rich Text Format，富文本格式）是一种在应用程序之间的传输所使用的，对格式化文本和图片的编码方式。这种格式得到了 MS-DOS、Windows、OS/2 和 Macintosh 平台上各种字处理程序的广泛支持	rtf	
text/sgml	SGML 标记文件	sgml、sgm	RFC 1874
text/t140	支持同步 RTP 多媒体中使用的标准 T.140 文本		RFC 2793
text/tab-separated-values	TSV 是在数据库和电子表格以及文字处理软件之间进行数据交换的通用方法。它由很多行构成，每行都有一些由 tab 字符分隔的字段	tsv	http://www.isi.edu/in-notes/iana/assignments/media-types/text/tab-separated-values
text/uri-list	URN 解析程序和所有需要进行大量 URI 列表通信的其他应用程序使用的简单的、经过注释的 URL 和 URN 列表	uris、uri	RFC 2483

566

（续）

MIME类型	描 述	扩 展	联系方式和参考文献
text/vnd.abc	ABC 文件是乐谱的可读格式	abc	http://www.gre.ac.uk/~c.walshaw/abc/ http://home1.swipnet.se/~w-11382/abcbnf.htm
text/vnd.curl	提供了一组由 CURL 运行时插件解释的内容定义语言	curl	Tim Hodge thodge@curl.com
text/vnd.DMClientScript	Dream Seeker 客户端应用程序会访问一些非 HTTP 站点（比如 BYOND、IRC 或 telnet），CommonDM Client Script 文件被当作指向这些站点的超链接使用	dms	Dan Bradley dan@dantom.com http://www.byond.com/code/ref/
text/vnd.fly	Fly 是个文本预处理器，它会用简单的语法来创建数据库和 Web 页面之间的接口	fly	John-Mark Gurney jmg@flyidea.com http://www.flyidea.com
text/vnd.fmi.flexstor	用于 SUVDAMA 和 UVRAPPF 项目	flx	http://www.ozone.fmi.fi/SUVDAMA/ http://www.ozone.fmi.fi/UVRAPPF/
text/vnd.in3d.3dml	用于 In3D 播放器	3dml、3dm	Michael Powers powers@insideout.net
text/vnd.in3d.spot	用于 In3D 播放器	spot、spo	同上
text/vnd.IPTC.NewsML	IPTC 国际新闻电信委员会指定的 NewsML 格式	xml	David Allen m_director_iptc@dial.pipex.com http://www.iptc.org
text/vnd.IPTC.NITF	IPTC 指定的 NITF 格式	xml	同上 http://www.nitf.org
text/vnd.latex-z	支持包含 Z 表示法的 LaTeX 文档。Z 表示法（读作“zed”）是基于 Zermelo-Fraenkel 集合论和一阶谓词逻辑的，有助于描述计算机系统		http://www.comlab.ox.ac.uk/archive/z/
text/vnd.motorola.reflex	提供了一种从 ReFLEX ™无线设备提交简单文本的通用的方式		Mark Patton fmp014@email.mot.com 有许可协议的情况下，可以从摩托罗拉得到（Enabling Protocol specification）的 FLEXsuite ™部分

（续）

MIME类型	描　　述	扩　　展	联系方式和参考文献
text/vnd.ms-mediapackage	这种类型由微软应用程序 MStore.exe 和 7 storDB.exe 处理	mpf	Jan Nelson jann@microsoft.com
text/vnd.wap.si	SI（Service Indication，业务指示符）对象中包含了用于描述事件的报文和用于描述从何处装载相应服务的 URI	si、xml	WAP Forum Ltd http://www.wapforum.org
text/vnd.wap.sl	SL（Service Loading，服务装载）内容类型提供了一种向移动客户端的用户代理传送 URI 的手段。客户端自身会自动地装载 URI 所指向的内容，在适当的时候且没有用户干预的情况下，在指定的用户代理上执行它	sl、xml	同上
text/vnd.wap.wml	WML（Wireless Markup Language，无线标记语言）是一种基于 XML 的标记语言，定义了包含移动电话和寻呼机在内的窄带设备的内容和用户接口	wml	同上
text/vnd.wap.wmlscript	WMLScript 是一种用于无线设备的 JavaScript 的拓展	wmls	同上
text/x-setext	来自 Apache 的 mime.types	etx	
text/xml	Extensible Markup 语言格式文件（如果下载时希望浏览器将文件保存下来就使用 application/xml）	xml	RFC 2376

567

D.4.9　video/*

表 D-11 列出了一些常见的视频电影格式。注意，有些视频格式被划分为 application 类型。

表D-11　“Video”MIME类型

MIME类型	描　　述	扩　　展	联系方式和参考文献
video/MP4V-ES	RTP 承载的 MPEG-4 视频负载		RFC 3016
video/mpeg	根据 ISO 11172 CD MPEG 标准编码的视频	mpeg、mpg、mpe	RFC 1341
video/parityfec	RTP 流上承载的数据所使用的前向纠错视频格式		RFC 3009
video/pointer	传输演示指针的位置信息		RFC 2862
video/quicktime	苹果 Quicktime 的视频格式	qt、mov	http://www.apple.com

（续）

MIME类型	描　　述	扩　　展	联系方式和参考文献
video/vnd.fvt	FAST Search & Transfer 公司的视频格式	fvt	Arild Fuldseth Arild.Fuldseth@fast.no 568
video/vnd.motorola.videovideo/vnd.motorola.videop	摩托罗拉 ISG 产品的特有格式		Tom McGinty Motorola ISG tmcginty@dma.isg.mot
video/vnd.mpegurl	这种媒体类型由一系列 MPEG 视频文件的 URL 构成	mxu	Heiko Recktenwald uzs106@uni-bonn.de “Power and Responsibility: Conversations with Contributors”, Guy van Belle 等，LMJ 9 (1999)，127-133，129 (MIT Press)
video/vnd.nokia.interleaved-multimedia	在诺基亚 9210 Communicator 视频播放器及相关工具中使用	nim	Petteri Kangaslampi petteri.kangaslampi@nokia.com
video/x-msvideo	微软的 AVI 电影	avi	http://www.microsoft.com
video/x-sgi-movie	硅图公司的电影格式	movie	http://www.sgi.com

D.4.10　实验类型

主类型集支持大多数内容类型。表 D-12 列出了一种实验类型，用于在某些 Web 服务器上配置的会议软件。

表D-12　扩展MIME类型

MIME类型	描　　述	扩　　展	联系方式和参考文献
x-conference/x-cooltalk	网景公司的协作工具	ice	569

附录E

Base-64编码

HTTP 将 Base-64 编码用于基本认证及摘要认证，在几种 HTTP 扩展中也使用了该编码。本附录解释了 Base-64 编码，提供了转换表和指向 Perl 软件的指针，可以帮助你在 HTTP 软件中正确使用 Base-64 编码。

E.1 Base-64编码保证了二进制数据的安全

Base-64 编码可以将任意一组字节转换成较长的常见文本字符序列，从而可以合法地作为首部字段值。Base-64 编码将用户输入或二进制数据，打包成一种安全格式，将其作为 HTTP 首部字段的值发送出去，而无须担心其中包含会破坏 HTTP 分析程序的冒号、换行符或二进制值。

Base-64 编码是作为 MIME 多媒体电子邮件标准的一部分开发的，这样 MIME 就可以在不同的合法电子邮件网关之间传输富文本和任意的二进制数据了。[1]Base-64 编码与将二进制数据文本化表示的 uuencode 和 BinHex 标准在本质上很类似，但空间效率更高。MIME RFC 2045 的第 6.8 节详细介绍了 Base-64 算法。

E.2 8位到6位

Base-64 编码将一个 8 位字节序列拆散为 6 位的片段，并为每个 6 位的片段分配一个字符，这个字符是 Base-64 字母表中的 64 个字符之一。这 64 个输出字符都是很
570 常见的，可以安全地放在 HTTP 首部字段中。这 64 个字符中包含大小写字母、数字、+ 和 /，还使用了特殊字符 =。表 E-1 显示了 Base-64 的字母表。

注意，由于 Base-64 编码用了 8 位字符来表示信息中的 6 个位，所以 Base-64 编码字符串大约比原始值扩大了 33%。

表E-1 Base-64字母表

0	A	8	I	16	Q	24	Y	32	g	40	o	48	w	56	4
1	B	9	J	17	R	25	Z	33	h	41	p	49	x	57	5
2	C	10	K	18	S	26	a	34	i	42	q	50	y	58	6
3	D	11	L	19	T	27	b	35	j	43	r	51	z	59	7
4	E	12	M	20	U	28	c	36	k	44	s	52	0	60	8
5	F	13	N	21	V	29	d	37	l	45	t	53	1	61	9
6	G	14	O	22	W	30	e	38	m	46	u	54	2	62	+
7	H	15	P	23	X	31	f	39	n	47	v	55	3	63	/

注 1：有些邮件网关会悄悄地去除 ASCII 值在 0 ～ 31 之间的“非打印”字符。其他程序会将一些字节作为流量控制字符或其他特殊控制字符来解释，或将回车符转换成换行符之类的字符。有些程序在收到带有值大于 127 的国际字符时会出现致命的错误，因为其软件不是“8 位干净”（8-bitclean）的。

图 E-1 是一个简单的 Base-64 编码实例。在这里，三个字符组成的输入值“Ow!”是 Base-64 编码的，得到的是 4 个字符的 Base-64 编码值“T3ch”。它是按以下方式工作的。

(1) 字符串“Ow!”被拆分成 3 个 8 位的字节（0x4F、0x77、0x21）。

(2) 这 3 个字节构成了一个 24 位的二进制值 010011110111011100100001。

(3) 这些位被划分为一些 6 位的序列 010011、110111、01110、100001。

(4) 每个 6 位值都表示了从 0 ～ 63 之间的一个数字，对应 Base-64 字母表中 64 个字符之一。得到的 Base-64 编码字符串是个 4 字符的字符串“T3ch”，然后就可以通过线路将这个字符串作为“安全的”8 位字符传送出去，因为只用了一些移植性最好的字符（字母、数字等）。

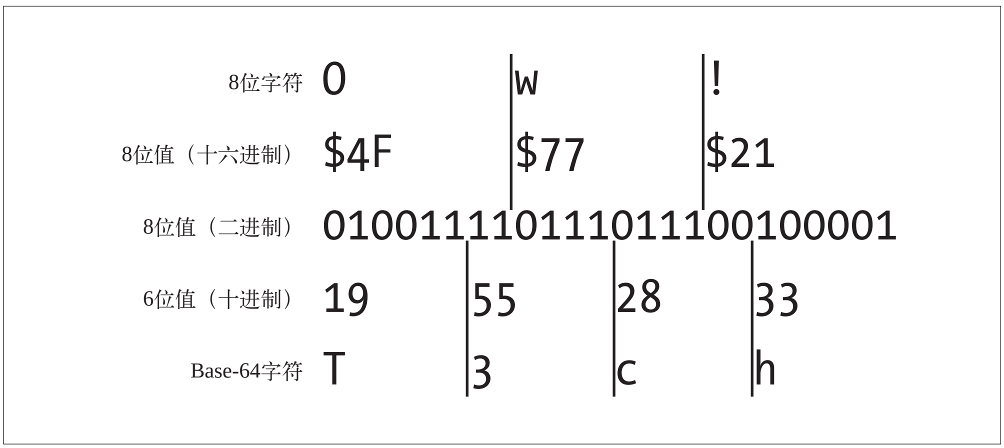


图 E-1　Base-64 编码实例 571

E.3　Base-64填充

Base-64 编码收到一个 8 位字节序列，将这个二进制序列流划分成 6 位的块。二进制序列有时不能正好平均地分成 6 位的块，在这种情况下，就在序列末尾填充零位，使二进制序列的长度成为 24 的倍数（6 和 8 的最小公倍数）。

对已填充的二进制串进行编码时，任何完全填充（不包含原始数据中的位）的 6 位组都由特殊的第 65 个符号“=”表示。如果 6 位组是部分填充的，就将填充位设置为 0。

表 E-2 显示了一些填充实例。初始输入字符串“a:a”为 3 字节（24 位）。24 是 6 和 8 的倍数，因此无需填充，得到的 Base-64 编码字符串为“YTph”。

表E-2　Base-64填充实例

输入数据	二进制序列（填充位以x表示）	已编码数据
a:a	011000 010011 101001 100001	YTph
a:aa	011000 010011 101001 100001 011000 01xxxx xxxxxx xxxxxx	YTphYQ==
a:aaa	011000 010011 101001 100001 011000 010110 0001xx xxxxxx	YTphYWE=
a:aaaa	011000 010011 101001 100001 011000 010110 000101 100001	YTphYWFh

然而，再增加一个字符，输入字符串会变成 32 位长。而 6 和 8 的下一个公倍数是 48，因此要添加 16 位的填充码。填充的前 4 位是与数据位混合在一起的。得到的 6 位组 01xxxx，会被当作 010000、十进制中的 16，或者 Base-64 编码的 Q 来处理。剩下的两个 6 位组都是填充码，用“=”表示。

E.4　Perl实现

MIME::Base64 是 Perl 中的 Base-64 编 / 解码模块。可以在 http://www.perldoc.com/perl5.6.1/lib/MIME/Base64.html 上看到有关这个模块的内容。

572 可以用 MIME::Base64 encode_base64 和 decode_base64 方法对字符串进行编解码：

```
use MIME::Base64;

$encoded = encode_base64('Aladdin:open sesame');
$decoded = decode_base64($encoded);
```

E.5　更多信息

更多有关 Base-64 编码的信息，参见以下信息。

http://www.ietf.org/rfc/rfc2045.txt
RFC 2045 的第 6.8 节，“MIME Part 1: Format of Internet Message Bodies,”（MIME 的第一部分：因特网报文主体的格式），是 Base-64 编码的官方规范。

http://www.perldoc.com/perl5.6.1/lib/MIME/Base64.html
这个 Web 站点提供了对 Base-64 字符串进行编 / 解码的 MIME::Base64 Perl 模块的
573 文档。

附录F

摘要认证

本附录包含了实现 HTTP 摘要认证功能所需的支撑数据和源代码。

F.1 摘要WWW-Authenticate指令

表 F-1 根据 RFC 2617 中的描述，对 WWW-Authenticate 指令进行了说明。与往常一样，最新细节请参见官方规范。

表F-1 （来自RFC 2617的）摘要WWW-Authenticate首部指令

指　令	描　述
realm	显示给用户的字符串，这样用户就可以知道该使用哪个用户名和密码了。这个字符串中至少应该包含执行认证功能的主机名字，此外可能还会说明可能拥有访问权的用户的集合。例如，registered_users@gotham.news.com
nonce	服务器特有的数据字符串，每次产生一个 401 响应时都应该生成一个唯一的数据字符串。建议这个字符串为 Base-64 或十六进制数据。需要特别说明的是，由于此字符串是放在首部行中作为引用字符串传送的，所以不允许使用双引号。 nonce 的内容是与实现有关的。实现的质量取决于选择是否合适。比如，可以将 nonce 构造成以下内容的 Base-64 编码： `time-stamp H(time-stamp ":" ETag ":" private-key)` 其中 time-stamp 是服务器生成的时间或其他不重复的数值，ETag 是与所请求实体有关的 HTTP ETag 首部的值，private-key 是只有服务器知道的数据。使用这种形式的 nonce，服务器会在收到客户端的 Authentication 首部之后重新对散列部分进行计算，如果与该首部的 nonce 不符，或者时间戳的值不够近，就可以拒绝请求。通过这种方式，服务器可以限制 nonce 的有效时间。包含 ETag 可以防止对资源更新版本的重放请求。（注意：在 nonce 中包含客户端的 IP 地址，看起来好像为服务器提供了限制最初获得此 nonce 的客户端重用 nonce 的能力，但这样会破坏代理集群，来自单个用户的请求通常都会经过集群中不同的代理进行传输。而且，IP 地址欺骗也不是很难。） 实现可以选择不接受以前用过的 nonce，或以前用过的摘要，以防止重放攻击，或者选择为 POST 或 PUT 请求使用一次性 nonce 或摘要，为 GET 请求使用时间戳 574
domain	一个引用的、由空格分隔的 URI 列表（如 RFC 2396，“Uniform Resource Identifiers: GenericSyntax”所述），这些 URI 定义了保护空间。如果 URI 是个 abs_path，它就是相对于受访服务器的典型根 URL 的。这个列表中的绝对 URI 所指的服务器可能不是受访服务器。 客户端可以用这个列表来判定应该将同样的认证信息发送给哪个 URI 集：可以假定所有以此列表中的 URI 作为前缀的 URI（在将两者都转换为绝对 URI 之后）都位于同一个保护空间内。 如果省略了这条指令，或者其值为空，客户端就应该假定保护空间中包含了响应服务器上的所有 URI。 这条指令在 Proxy-Authenticate 首部是无意义的，此时，保护空间总是包括整个代理；如果提供了这条指令，也应该将其忽略

（续）

指　　令	描　　述
opaque	一个由服务器指定的数据串，应该由客户端不经修改地放在后继请求的 `Authorization` 首部中返回，这些后继请求应使用同一保护空间内的 URI。建议这个字符串采用 Base-64 或十六进制的数据
stale	一个标志，用来说明由于 nonce 值太过陈旧，前一条来自客户端的请求被拒绝了。如果 stale 为 TRUE（不区分大小写），客户端可能希望以新加密的响应重试请求，而不用再次提示用户输入新的用户名和密码。只有在服务器收到一条 nonce 无效，但摘要有效的请求（说明客户端知道正确的用户名 / 密码）时，才应该将 stale 设置为 TRUE。如果 stale 为 FALSE，或者除 TRUE 之外的其他值，或者没有提供 stale 指令，用户名和 / 或密码就是无效的，需要获取新的值

algorithm	一个字符串，说明了一对儿用来生成摘要和校验码的算法。如果没有提供这个字符串，就假定它为“MD5”。如果不识别此算法，就忽略这种质询（如果有多个算法的话，就使用另外一个）。
	在这份文档中，用“KD(secret,data)”来表示用密码“secret”对数据“data”使用摘要算法得到的字符串，而对数据“data”使用校验和算法得到的字符串则表示为“H(data)”。表示法“unq(X)”表示引用字符串“X”的值（不包含左右两边的引号）。
	对 MD5 和 MD5-sess 算法来说：
	`H(data) = MD5(data)`
	`HD(secret, data) = H(concat(secret, ":", data))`
	也就是说，摘要就是将密码的 MD5 与冒号和数据连接在一起。MD5-sess 算法目的是支持使用高效的第三方认证服务器

qop	这条指令是可选的，只是为了与 RFC 2069[6] 后向兼容才保留的。所有与此版本的摘要方案兼容的实现都应该使用它。 如果提供了这条指令，它就是由一个或多个标记构成的引用字符串，用来说明服务器所支持的“安全保障”值。值 auth 说明要进行认证，值 auth-int 说明要进行具有完整性保护的认证。一定要忽略那些不识别的选项
<extension>	未来可以通过这条指令进行扩展。要忽略所有不认识的指令

F.2　摘要Authorization指令

表 F-2 根据 RFC 2617 的描述，对每条摘要 `Authorization` 指令都进行了说明。最新的细节请参见官方规范。 575

表F-2　（来自RFC 2617的）摘要Authorization首部指令

指　　令	描　　述
username	指定域中的用户名
realm	在 `WWW-Authenticate` 首部中传送给客户端的域

（续）

指　　令	描　　述
nonce	在 WWW-Authenticate 首部中传送给客户端的那个与服务器相同的 nonce
uri	来自请求行请求 URI 中的 URI。由于代理可以在传输中修改请求行，所以会出现重复。进行正确的摘要验证计算可能需要原始 URI
response	这个就是实际的摘要——摘要认证的重点！响应是一个由 32 个十六进制数字组成的字符串，由沟通好的摘要算法生成，用来证明用户知道这个密码
algorithm	一个字符串，说明了用来生成摘要和校验和的一对儿算法。如果未提供，就将其假定为 MD5
opaque	服务器在 WWW-Authenticate 首部指定的数据串，应该由客户端不经修改地在后继请求的 WWW-Authenticate 首部中返回，这个请求应使用同一保护空间内的 URI
cnonce	如果发送了 qop 指令，就一定要使用这条指令，如果服务器没有在 WWW-Authenticate 首部字段中发送 qop 指令，就一定不能使用这条指令。 cnonce 值是客户端提供的不透明引用字符串值，客户端和服务器都用它来避免选择明文攻击，以提供双向认证以及一些报文一致性检查。 参见本附录稍后介绍的响应摘要和请求摘要计算
qop	说明客户端对报文应用的“安全保障”是什么。如果提供了这条指令，它的值就必须是服务器在 WWW-Authenticate 首部中说明的、它支持的可选值之一。这些值会影响请求摘要的计算方式。 它只是个单独的标记，而不是 WWW-Authenticate 中那样的可选值引用列表。 这条指令是可选的，以保持对 RFC 2069 最小实现的后向兼容，但如果服务器说明它是通过在 WWW-Authenticate 首部字段中提供 qop 指令来支持 qop 的，就应该使用这条指令
nc	如果发送了 qop 指令，就一定要指定这条指令，如果服务器没有在 WWW-Authenticate 首部字段中发送 qop 指令，就一定不能使用这条指令。 其值是个十六进制值，表示客户端已经发送的包含 nounce 值的请求次数（包括当前请求在内）。比如，作为对指定 nonce 值的响应发送的第一条请求中，客户端会发送 nc="00000001"。 这条指令的目的是允许服务器通过维护自己保存的此计数值副本来发现请求重放——如果看到了两次相同的 nc 值，就说明请求是重放的
<extension>	未来可以通过这条指令进行扩展。所有不认识的指令都要忽略掉

F.3　摘要Authentication-Info指令

表 F-3 根据 RFC 2617 的描述，对每条 Authentication-Info 指令都进行了说明。
576 最新的细节请参见官方规范。

表F-3　（来自RFC 2617的）摘要`Authentication-Info`首部指令

指　　令	描　　述
nextnonce	nextnonce 指令的值是服务器希望客户端为未来的认证响应使用的 nonce。服务器可能会发送带有 nextnonce 字段的 `Authentication-Info` 首部，作为实现一次性 nonce 或修改 nonce 的手段。如果提供了 nextnonce 字段，客户端在为下一条请求构建 `Authorization` 首部时就应该使用它。客户端如果没能做到，就会收到来自服务器的 `"stale=TRUE"` 认证请求。 服务器实现应该仔细地考虑使用这种机制带来的性能影响。如果每条响应都包含了必须在服务器接收的下一条请求中使用的 nextnounce 指令，就不可能使用管道化请求了。应该考虑在性能和安全之间进行一些平衡，允许在有限的时间内使用老的 nonce 值，以实现请求的管道化。使用 nounce 计数可以在不影响管道化的情况下，维护一个新的服务器 nonce 的大部分安全优势
qop	说明了服务器应用到响应上的“安全保障”选项。值 auth 说明要进行认证，值 auth-int 说明要进行带有完整性保护的认证。服务器在响应中使用的 qop 指令值应该与客户端在相应请求中发送的值相同

rspauth	response auth 指令中的可选响应摘要支持双向认证——服务器证明了它知道用户的密码，而且通过 `qop="auth-int"`，它还为响应提供了有限的完整性保护。除了当 `qop="auth"` 或者没有在 `Authorization` 首部为请求指定 qop 的情况，`response-digest` 值的计算方式与 `Authorization` 首部的 `request-digest` 类似，A2 为：
	`A2 = ":" digest-uri-value`
	当 `qop="auth-int"` 时，A2 为：
	`A2 = ":" digest-uri-value ":" H(entity-body)`
	其中 `digest-uri-value` 是请求的 `Authorization` 首部中 uri 指令的值。cnonce 和 nc 值一定要与此报文所响应的客户端请求中的相应值相同。如果指定了 `qop="auth "` 或者 `qop="auth-int"`，就必须提供 rspauth 指令

cnonce	cnonce 值一定要与此报文所响应的客户端请求中的相应值一样。如果指定了 `qop="auth "` 或 `qop="auth-int"`，就必须提供 cnonce 指令
nc	nc 值一定要与此报文所响应的客户端请求中的相应值一样。如果指定了 `qop="auth "` 或 `qop="auth-int"`，就必须提供 nc 指令
<extension>	未来可以通过这条指令进行扩展。所有不识别的指令都要忽略掉

F.4　参考代码

下列代码实现了 RFC 2617 中 H(A1)、H(A2)、request-digest 和 response-digest 的计算。它使用了 RFC 1321 中的 MD5 实现。

F.4.1　文件digcalc.h

```
#define HASHLEN 16
typedef char HASH[HASHLEN];
#define HASHHEXLEN 32
typedef char HASHHEX[HASHHEXLEN+1];
577 #define IN
#define OUT
/* calculate H(A1) as per HTTP Digest spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    );

/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1, /* H(A1) */
    IN char * pszNonce, /* nonce from server */
    IN char * pszNonceCount, /* 8 hex digits */
    IN char * pszCNonce, /* client nonce */
    IN char * pszQop, /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod, /* method from the request */
    IN char * pszDigestUri, /* requested URL */
    IN HASHHEX HEntity, /* H(entity body) if qop="auth-int" */
    OUT HASHHEX Response /* request-digest or response-digest */
    );
```

F.4.2　文件“digcalc.c”

```
#include <global.h>
#include <md5.h>
#include <string.h>
#include "digcalc.h"

void CvtHex(
    IN HASH Bin,
    OUT HASHHEX Hex
    )
    {

unsigned short i;
unsigned char j;
for (i = 0; i < HASHLEN; i++) {
    j = (Bin[i] >> 4) & 0xf;
    if (j <= 9)
        Hex[i*2] = (j + '0');
    else
        Hex[i*2] = (j + 'a' - 10);
    j = Bin[i] & 0xf;
```

```
    if (j <= 9)
        Hex[i*2+1] = (j + '0');
    else
        Hex[i*2+1] = (j + 'a' - 10);
    };
    Hex[HASHHEXLEN] = '\0';
    };
```
578

```
/* calculate H(A1) as per spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    )
{
    MD5_CTX Md5Ctx;
    HASH HA1;
    MD5Init(&Md5Ctx);
    MD5Update(&Md5Ctx, pszUserName, strlen(pszUserName));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszRealm, strlen(pszRealm));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszPassword, strlen(pszPassword));
    MD5Final(HA1, &Md5Ctx);
    if (stricmp(pszAlg, "md5-sess") == 0) {
        MD5Init(&Md5Ctx);
        MD5Update(&Md5Ctx, HA1, HASHLEN);
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
        MD5Final(HA1, &Md5Ctx);
    };
    CvtHex(HA1, SessionKey);
};
/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1, /* H(A1) */
    IN char * pszNonce, /* nonce from server */
    IN char * pszNonceCount, /* 8 hex digits */
    IN char * pszCNonce, /* client nonce */
    IN char * pszQop, /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod, /* method from the request */
    IN char * pszDigestUri, /* requested URL */
    IN HASHHEX HEntity, /* H(entity body) if qop= "auth-int" */
    OUT HASHHEX Response /* request-digest or response-digest */
    )
{
    MD5_CTX Md5Ctx;
    HASH HA2;
    HASH RespHash;
    HASHHEX HA2Hex;
    // calculate H(A2)
```

```
        MD5Init(&Md5Ctx);
        MD5Update(&Md5Ctx, pszMethod, strlen(pszMethod));
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszDigestUri, strlen(pszDigestUri));
        if (stricmp(pszQop, "auth-int") == 0) {
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, HEntity, HASHHEXLEN);
    };
    MD5Final(HA2, &Md5Ctx);
    CvtHex(HA2, HA2Hex);
    // calculate response
    MD5Init(&Md5Ctx);
    MD5Update(&Md5Ctx, HA1, HASHHEXLEN);
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
    MD5Update(&Md5Ctx, ":", 1);
    if (*pszQop) {
           MD5Update(&Md5Ctx, pszNonceCount, strlen(pszNonceCount));
           MD5Update(&Md5Ctx, ":", 1);
           MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
           MD5Update(&Md5Ctx, ":", 1);
           MD5Update(&Md5Ctx, pszQop, strlen(pszQop));
           MD5Update(&Md5Ctx, ":", 1);
        };
        MD5Update(&Md5Ctx, HA2Hex, HASHHEXLEN);
        MD5Final(RespHash, &Md5Ctx);
        CvtHex(RespHash, Response);
    };
```

F.4.3 文件digtest.c

```
#include <stdio.h>
#include "digcalc.h"

void main(int argc, char ** argv) {
    char * pszNonce = "dcd98b7102dd2f0e8b11d0f600bfb0c093";
    char * pszCNonce = "0a4f113b";
    char * pszUser = "Mufasa";
    char * pszRealm = "testrealm@host.com";
    char * pszPass = "Circle Of Life";
    char * pszAlg = "md5";
    char szNonceCount[9] = "00000001";
    char * pszMethod = "GET";
    char * pszQop = "auth";
    char * pszURI = "/dir/index.html";
    HASHHEX HA1;
    HASHHEX HA2 = "";
    HASHHEX Response;
    DigestCalcHA1(pszAlg, pszUser, pszRealm, pszPass,
        pszNonce, pszCNonce, HA1);
        DigestCalcResponse(HA1, pszNonce, szNonceCount, pszCNonce, pszQop,
        pszMethod, pszURI, HA2, Response);
    printf("Response = %s\n", Response);
};
```

579～580

附录G

语言标记

语言标记是一些短小的标准字符串，用来命名所使用的语言——比如，fr（法语）和 en-GB（英式英语）。每个标记都由一个或多个称为子标记的部分组成，中间由连字符分隔。16.4 节曾详细介绍了语言标记。

本附录总结了语言标记的规则、标准化的标记以及注册信息。包含下列参考资料：

- G.1 节总结了第一个（主）子标记所用规则；
- G.2 节总结了第二个子标记所用规则；
- 表 G-1 显示了已在 IANA 注册的语言标记；
- 表 G-2 列出了 ISO 639 语言代码；
- 表 G-3 列出了 ISO 3166 国家代码。

G.1 第一个子标记所用规则

如果第一个子标记为：

- 2 个字符长，是 ISO 639[1] 和 639-1 标准的语言代码；
- 3 个字符长，是 ISO 639-2[2] 标准中列出的语言代码；
- 字母“i”，语言标记就是在 IANA 中显式注册过的语言标记；
- 字母“x”，语言标记就是私有的、非标准的扩展子标记。

581 表 G-2 总结了 ISO 639 和 639-2 的名称。

G.2 第二个子标记所用规则

如果第二个子标记为：

- 2 个字符长，就是 ISO 3166[3] 定义的国家 / 地区码；
- 3 ～ 8 个字符长，是已在 IANA 注册过的语言标记；
- 1 个字符长，就是非法的。

表 G-3 总结了 ISO 3166 国家代码。

注 1：参见 ISO 标准 639，“Codes for the representation of names of languages”。

注 2：参见 ISO 639-2，“Codes for the representation of names of languages—Part 2: Alpha-3 code”（语言名表示码——第 2 部分：Alpha-3 代码）。

注 3：ISO 3166 将国家代码 AA、QM-QZ、XA-XZ 和 ZZ 保留作为用户分配的代码。不能用它们来构造语言标记。

G.3 IANA已注册的语言标记

表G-1 语言标记

IANA语言标记	描　述
i-bnn	布农语
i-default	默认语言背景
i-hak	客家语
i-klingon	克林根语
i-lux	卢森堡语
i-mingo	明戈语
i-navajo	纳瓦霍语
i-pwn	排湾语
i-tao	越南语
i-tay	泰雅语
i-tsu	邹族语
no-bok	挪威“书面语言”
no-nyn	挪威“新挪威语”
zh-gan	赣语
zh-guoyu	普通话或标准汉语
zh-hakka	客家话
zh-min	闽语、福州话、福建话、厦门话或台湾话
zh-wuu	上海话或吴语
zh-xiang	湘语或湖南话
zh-yue	粤语

582

G.4 ISO 639语言代码

表G-2 ISO 639和ISO 639-2语言代码

语　言	ISO 639	ISO 639–2
阿布哈西亚语	ab	abk
亚齐语		ace
阿乔利语		ach
阿当梅语		ada
阿法尔语	aa	aar

（续）

语　　言	ISO 639	ISO 639-2
阿弗里希利语		afh
南非语	af	afr
其他亚非语系		afa
阿坎语		aka
阿卡德语		akk
阿尔巴尼亚语	sq	alb/sqi
阿留申语		ale
阿尔冈昆语系		alg
其他阿尔泰语系		tut
阿姆哈拉语	am	amh
阿帕切语		apa
阿拉伯语	ar	ara
阿拉米语		arc
阿拉帕霍语		arp
阿劳卡尼亚语		arn
阿拉瓦克语		arw
亚美尼亚语	hy	arm/hye
其他人工语言		art
阿萨姆语	as	asm
阿撒帕斯坎巴斯卡语系		ath
其他南岛语系		map
阿瓦尔语		ava
阿维斯陀语		ave
阿瓦德语		awa
艾马拉语	ay	aym
阿塞拜疆语	az	aze
阿兹台克语		nah
巴厘语		ban
583 其他波罗地语系语言		bat
俾路支语		bal
班巴拉语		bam

（续）

语　　言	ISO 639	ISO 639-2
巴米累克语		bai
班达语		bad
其他班图语系语言		bnt
巴萨语		bas
巴什基尔语	ba	bak
巴斯克语	eu	baq/eus
贝雅语		bej
本巴语		bem
孟加拉语	bn	ben
其他帕帕尔语		ber
博杰普尔语		bho
比哈尔语	bh	bih
比科尔语		bik
比尼语		bin
比斯拉马语	bi	bis
布拉吉语		bra
布尔吞语	be	bre
布吉语		bug
保加利亚语	bg	bul
布里亚特语		bua
缅甸语	my	bur/mya
白俄罗斯语	be	bel
卡多语		cad
巴勒比语		car
加泰罗尼亚语	ca	cat
其他高加索语系语言		cau
宿务语		ceb
其他凯尔特语系语言		cel
其他中美洲印第安语系语言		cai
查加台语		chg
查莫罗语		cha

（续）

584

语　　言	ISO 639	ISO 639-2
车臣语		che
切罗基语		chr
夏延语		chy
契布卡语		chb
汉语	zh	chi/zho
契努克混合语		chn
乔克托语		cho
古教会斯拉夫语		chu
楚瓦什语		chv
科普特语		cop
康沃尔语		cor
科西嘉语	co	cos
克里语		cre
克里克语		mus
其他克里奥尔混合语		crp
其他英语克里奥尔混合语		cpe
其他法语克里奥尔混合语		cpf
其他葡萄牙语克里奥尔混合语		cpp
其他库什特语系语言		cus
克罗地亚语	hr	
捷克语	cs	ces/cze
达科他语		dak
丹麦语	da	dan
特拉华语		del
丁卡语		din
迪维希语		div
多格拉语		doi
其他德拉维语系语言		dra
杜亚拉语		dua
荷兰语	nl	dut/nla

（续）

语　　言	ISO 639	ISO 639-2	
中古荷兰语（约 1050 年～ 1350 年）		dum	
迪乌拉语		dyu	
宗喀语	dz	dzo	
艾菲克语		efi	
古埃及语		egy	
埃克丘克语		eka	
埃兰语		elx	
英语	en	eng	
中古英语（约 1100 年～ 1500 年）		enm	585
古英语（约 450 年～ 1100 年）		ang	
其他爱斯基摩语系语言		esk	
世界语	eo	epo	
爱沙尼亚语	et	est	
爱威语		ewe	
艾旺多语		ewo	
芳语		fan	
芳蒂语		fat	
法罗语	fo	fao	
斐济语	fj	fij	
芬兰语	fi	fin	
其他芬兰－乌戈尔语系语言		fiu	
丰族语		fon	
法语	fr	fra/fre	
中古法语（约 1400 年～ 1600 年）		frm	
古法语（约 842 年～ 1400 年）		fro	
弗里西亚语	fy	fry	
富拉语		ful	
加语		gaa	
盖尔语（苏格兰）		gae/gdh	
加利西亚语	gl	glg	
干达语		lug	

（续）

语　　言	ISO 639	ISO 639-2
卡约语		gay
古兹语		gez
格鲁吉亚语	ka	geo/kat
德语	de	deu/ger
中古高地德语（约 1050 年～ 1500 年）		gmh
古高地德语（约 750 年～ 1050 年）		goh
其他德语语系语言		gem
吉尔伯特语		gil
贡德语		gon
哥特语		got
格列博语		grb
古希腊语（至 1453 年）		grc
现代希腊语（1453 年至今）	el	ell/gre
586 格陵兰语	kl	kal
瓜拉尼语	gn	grn
古吉拉特语	gu	guj
海达语		hai
豪萨语	ha	hau
夏威夷语		haw
希伯来语	he	heb
赫雷罗语		her
希利盖农语		hil
喜马偕尔语		him
印地语	hi	hin
希里莫图语		hmo
匈牙利语	hu	hun
胡帕语		hup
伊班语		iba
冰岛语	is	ice/isl
伊博语		ibo

（续）

语　　言	ISO 639	ISO 639-2
伊乔语		ijo
伊洛卡诺语		ilo
其他印度 - 雅利安语系		inc
其他印欧语系语言		ine
印度尼西亚语	id	ind
国际语	ia	ina
西方国际语	ie	ine
因纽特语	iu	iku
伊努皮亚克语	ik	ipk
其他伊朗语系语言		ira
爱尔兰语	ga	gai/iri
古爱尔兰语（至 900 年）		sga
中古爱尔兰语（900 年～ 1200 年）		mga
伊洛魁语系语言		iro
意大利语	it	ita
日语	ja	jpn
爪哇语	jv/jw	jav/jaw
犹太阿拉伯语		jrb
犹太波斯语		jpr
卡布列语		kab
景颇语		kac
坎巴语		kam
卡纳达语	kn	kan
卡努里语		kau
卡拉 - 卡尔帕克语		kaa
克伦语		kar
克什米尔语	ks	kas
卡威语		kaw
哈萨克语	kk	kaz
卡西语		kha
高棉语	km	khm

587

（续）

语　　言	ISO 639	ISO 639-2
其他科依桑语系语言		khi
和阗语		kho
基库尤语		kik
卢旺达语	rw	kin
柯尔克孜语	ky	kir
科米语		kom
刚果语		kon
孔卡尼语		kok
韩语	ko	kor
克佩勒语		kpe
克鲁语		kro
宽亚玛语		kua
库梅克语		kum
库尔德语	ku	kur
库鲁克语		kru
克萨语		kus
库特内语		kut
拉迪诺语		lad
拉亨达语		lah
兰巴语		lam
奥克语（公元前 1500 年）	oc	oci
老挝语	lo	lao
拉丁语	la	lat
拉脱维亚语	lv	lav
卢森堡语		ltz
列兹基语		lez
林加拉语	ln	lin
立陶宛语	lt	lit
洛齐语		loz
卢巴－卡丹加语		lub
卢伊塞诺语		lui

588

（续）

语　　言	ISO 639	ISO 639–2
隆达语		lun
卢奥语（肯尼亚与坦桑比亚）		luo
马其顿语	mk	mac/mak
马都拉语		mad
摩揭陀语		mag
迈蒂利语		mai
旺加锡语		mak
马达加斯语	mg	mlg
马来语	ms	may/msa
马拉雅拉姆语		mal
马耳他语	ml	mlt
曼丁哥语		man
曼尼普尔语		mni
马诺博诸语		mno
曼岛语		max
毛利语	mi	mao/mri
马拉地语	mr	mar
马里语		chm
马绍尔语		mah
马尔瓦利语		mwr
马萨伊语		mas
玛雅语		myn
门德语		men
密克玛克语		mic
米南卡保语		min
多种其他语言		mis
莫霍克语		moh
摩尔达维亚语	mo	mol
其他孟高棉语族		mkh
芒戈语		lol
蒙古语	mn	mon

589

（续）

语　　言	ISO 639	ISO 639-2
默希语		mos
多语种		mul
蒙达语族		mun
瑙鲁语	na	nau
纳瓦霍语		nav
北恩德贝勒语		nde
南恩德贝勒语		nbl
恩敦加语		ndo
尼泊尔语	ne	nep
尼瓦里语		new
其他尼日尔－科尔多凡语系语言		nic
其他尼罗－撒哈拉语系		ssa
纽埃语		niu
古诺尔斯语		non
其他北美印第安语族语言		nai
挪威语	no	nor
挪威布克莫尔语		nno
努比亚语族		nub
尼扬韦齐语		nym
尼扬贾语		nya
尼扬科勒语		nyn
尼奥罗语		nyo
恩济马语		nzi
奥吉布瓦语		oji
奥利亚语	or	ori
奥罗莫语	om	orm
奥萨格语		osa
奥塞梯语		oss
奥托米安语系		oto
钵罗维语		pal
帕劳语		pau

（续）

语　　言	ISO 639	ISO 639-2	
巴利语		pli	
邦板牙语		pam	
邦阿西楠语		pag	
旁遮普语	pa	pan	590
帕皮亚门托语		pap	
其他巴布亚 - 澳洲语系语言		paa	
波斯语	fa	fas/per	
古波斯语（大约公元前 600 年～ 400 年）		peo	
腓尼基语		phn	
波兰语	pl	pol	
波纳佩语		pon	
葡萄牙语	pt	por	
普拉克里特语族		pra	
古普罗旺斯语（至 1500 年）		pro	
普什图语	ps	pus	
克丘亚语	qu	que	
里托罗曼斯语	rm	roh	
拉贾斯坦语		raj	
拉罗汤加语		rar	
其他罗曼语系语言		roa	
罗马尼亚语	ro	ron/rum	
罗姆语		rom	
基隆迪语	rn	run	
俄语	ru	rus	
萨利什语系		sal	
萨马利亚阿拉米语		sam	
萨米语系语言		smi	
萨摩亚语	sm	smo	
桑达韦语		sad	
桑戈语	sg	sag	

（续）

语　　言	ISO 639	ISO 639-2
梵语	sa	san
萨丁尼亚语		srd
苏格兰语		sco
塞尔库普语		sel
其他闪米特语系语言		sem
塞尔维亚语	sr	
塞尔维亚 - 克罗地亚语	sh	scr
塞雷尔语		srr
掸语		shn
591 修纳语	sn	sna
锡达莫语		sid
西克西卡语		bla
信德语	sd	snd
僧伽罗语	si	sin
其他汉藏语系语言		sit
苏语系语言		sio
其他斯拉夫语族		sla
西斯旺语	ss	ssw
斯洛伐克语	sk	slk/slo
斯洛文尼亚语	sl	slv
粟特语		sog
索马里语	so	som
桑海语		son
索布语系语言		wen
北索托语		nso
南索托语	st	sot
其他南美印第安语族语言		sai
西班牙语	es	esl/spa
苏库马语		suk
苏美尔语		sux
苏丹语	su	sun

（续）

语　　言	ISO 639	ISO 639-2
苏苏语		sus
斯瓦西里语	sw	swa
斯沃兹语		ssw
瑞典语	sv	sve/swe
叙利亚语		syr
塔加洛语	tl	tgl
塔希提语		tah
塔吉克语	tg	tgk
塔马舍克语		tmh
泰米尔语	ta	tam
塔塔尔语	tt	tat
泰卢固语	te	tel
特列纳语		ter
泰语	th	tha
藏语	bo	bod/tib
提格雷语		tig
提格里尼亚语	ti	tir
滕内语		tem
蒂夫语		tiv
特林吉特语		tli
汤加语（尼亚萨）	to	tog
东加语（汤加岛）		ton
特鲁克语		tru
钦西安语		tsi
宗加语	ts	tso
茨瓦纳语	tn	tsn
奇图姆布卡语		tum
土耳其语	tr	tur
奥斯曼土耳其语（约 1500 年～ 1928 年）		ota
土库曼语	tk	tuk
图瓦语		tyv

592

（续）

语　　言	ISO 639	ISO 639-2
特威语	tw	twi
乌加里特语		uga
维吾尔语	ug	uig
乌克兰语	uk	ukr
翁本杜语		umb
未确定的语言		und
乌尔都语	ur	urd
乌兹别克语	uz	uzb
瓦伊语		vai
文达语		ven
越南语	vi	vie
沃拉普克语	vo	vol
沃提克语		vot
瓦卡什语系语言		wak
瓦拉莫语		wal
瓦赖语		war
瓦肖语		was
威尔士语	cy	cym/wel
沃洛夫语	wo	wol
593 科萨语	xh	xho
雅库特语		sah
瑶语		yao
雅浦语		yap
依地语	yi	yid
约鲁巴语	yo	yor
萨波特克语		zap
哲纳加语		zen
壮语	za	zha
祖鲁语	zu	zul
祖尼语		zun

G.5 ISO 3166国家代码

表G-3 ISO 3166国家代码

国 家	代 码
阿富汗	AF
阿尔巴尼亚	AL
阿尔及利亚	DZ
美属萨摩亚	AS
安道尔	AD
安哥拉	AO
安圭拉	AI
南极洲	AQ
安提瓜和巴布达	AG
阿根廷	AR
亚美尼亚	AM
阿鲁巴	AW
澳大利亚	AU
奥地利	AT
阿塞拜疆	AZ
巴哈马	BS
巴林	BH
孟加拉	BD
巴巴多斯	BB
白俄罗斯	BY
比利时	BE
伯利兹	BZ
贝宁	BJ
百慕大	BM
不丹	BT
玻利维亚	BO
波黑	BA
博茨瓦纳	BW
布韦岛	BV

594

（续）

国　　家	代　　码
巴西	BR
英属印度洋领地	IO
文莱达鲁萨兰国	BN
保加利亚	BG
布基纳法索	BF
布隆迪	BI
柬埔寨	KH
喀麦隆	CM
加拿大	CA
佛得角	CV
开曼群岛	KY
中非共和国	CF
乍得	TD
智利	CL
中国	CN
圣诞岛	CX
科科斯（基林群岛）	CC
哥伦比亚	CO
科摩罗	KM
刚果	CG
刚果（金共和国）	CD
库克群岛	CK
哥斯达黎加	CR
科特迪瓦	CI
克罗地亚	HR
古巴	CU
塞浦路斯	CY
捷克	CZ
丹麦	DK
吉布提	DJ
多米尼克	DM

595

（续）

国　　家	代　　码
多米尼加共和国	DO
东帝汶	TP
厄瓜多尔	EC
埃及	EG
萨尔瓦多	SV
赤道几内亚	GQ
厄立特里亚	ER
爱沙尼亚	EE
埃塞俄比亚	ET
福克兰群岛（马尔维纳斯）	FK
法罗群岛	FO
斐济群岛	FJ
芬兰	FI
法国	FR
法属圭亚那	GF
法属波利尼西亚	PF
法国南部领地	TF
加蓬	GA
冈比亚	GM
格鲁吉亚	GE
德国	DE
加纳	GH
直布罗陀	GI
希腊	GR
格陵兰	GL
格林纳达	GD
瓜德罗普	GP
关岛	GU
危地马拉	GT
几内亚	GN
几内亚比绍	GW

（续）

国　　家	代　　码
圭亚那	GY
海地	HT
赫德岛和麦克唐纳群岛	HM
梵蒂冈	VA
洪都拉斯	HN
匈牙利	HU
冰岛	IS
印度	IN
印度尼西亚	ID
伊朗（伊斯兰共和国）	IR
伊拉克	IQ
爱尔兰	IE
以色列	IL
意大利	IT
牙买加	JM
日本	JP
约旦	JO
哈萨克斯坦	KZ
肯尼亚	KE
基里巴斯	KI
朝鲜（民主主义人民共和国）	KP
韩国（共和国）	KR
科威特	KW
吉尔吉斯斯坦	KG
老挝人民民主共和国	LA
拉脱维亚	LV
黎巴嫩	LB
莱索托	LS
利比里亚	LR
阿拉伯利比亚民众国	LY

（续）

国　　家	代　　码
列支敦士登	LI
立陶宛	LT
卢森堡	LU
马其顿（前南斯拉夫共和国）	MK
马达加斯加	MG
马拉维	MW
马来西亚	MY
马尔代夫	MV
马里	ML
马耳他	MT
马绍尔群岛	MH
马提尼克	MQ
毛里塔尼亚	MR
毛里求斯	MU
马约特	YT
墨西哥	MX
密克罗尼西亚（联邦）	FM
摩尔多瓦（共和国）	MD
摩纳哥	MC
蒙古国	MN
蒙塞拉特岛	MS
摩洛哥	MA
莫桑比克	MZ
缅甸	MM
纳米比亚	NA
瑙鲁	NR
尼泊尔	NP
荷兰	NL
荷属安的列斯	AN
新喀里多尼亚	NC
新西兰	NZ

597

（续）

国　　家	代　　码
尼加拉瓜	NI
尼日尔	NE
尼日利亚	NG
纽埃	NU
诺福克岛	NF
北马里亚那群岛	MP
挪威	NO
阿曼	OM
巴基斯坦	PK
帕劳	PW
巴勒斯坦领土（被占）	PS
巴拿马	PA
巴布亚新几内亚	PG
巴拉圭	PY
秘鲁	PE
菲律宾	PH
皮特凯恩	PN
波兰	PL
葡萄牙	PT
波多黎各	PR
卡塔尔	QA
留尼旺	RE
罗马尼亚	RO
俄罗斯	RU
卢旺达	RW
圣赫勒拿	SH
圣基斯和尼维斯	KN
圣卢西亚	LC
圣皮埃尔和密克隆	PM
圣文森特和格林纳丁斯	VC
萨摩亚	WS

598

（续）

国　　家	代　　码
圣马力诺	SM
圣多美和普林西比	ST
沙特阿拉伯	SA
塞内加尔	SN
塞舌尔	SC
塞拉利昂	SL
新加坡	SG
斯洛伐克	SK
斯洛文尼亚	SI
所罗门群岛	SB
索马里	SO
南非	ZA
南乔治亚岛和南桑威奇群岛	GS
西班牙	ES
斯里兰卡	LK
苏丹	SD
苏里南	SR
斯瓦尔巴群岛和扬马延岛	SJ
斯威士兰	SZ
瑞典	SE
瑞士	CH
阿拉伯叙利亚共和国	SY
塔吉克斯坦	TJ
坦桑尼亚（联合共和国）	TZ
泰国	TH
多哥	TG
托克劳	TK
汤加	TO
特立尼达和多巴哥	TT
突尼斯	TN
土耳其	TR

599

（续）

国　　家	代　　码
土库曼斯坦	TM
特克斯和凯科斯群岛	TC
图瓦卢	TV
乌干达	UG
乌克兰	UA
阿拉伯联合酋长国	AE
英国	GB
美国	US
美国本土外小岛屿	UM
乌拉圭	UY
乌兹别克斯坦	UZ
瓦努阿图	VU
委内瑞拉	VE
越南	VN
维尔京群岛（英属）	VG
维尔京群岛（美属）	VI
瓦利斯和富图纳	WF
西撒哈拉	EH
也门	YE
南斯拉夫	YU
600 赞比亚	ZM

G.6 语言管理组织

- ISO 639 指定了一个向 ISO 639 的语言列表中进行添加或者对其进行修改的维护机构。此机构为：
  International Information Centre for Terminology（Infoterm，国际术语信息中心）
  P.O. Box 130
  A-1021 Wien
  Austria
  电话：+43 1 26 75 35 分机号：312
  传真：+43 1 216 32 72

- ISO 639-2 指定了一个向 ISO 639-2 的语言列表中进行添加或者对其进行修改的维护机构。此机构为：

  Library of Congress（美国国会图书馆）
  Network Development and MARC Standards Office（网络开发与 MARC 标准办公室）
  Washington, D.C. 20540
  USA
  电话：+1 202 707 6237
  传真：+1 202 707 0115
  URL：http://www.loc.gov/standards/iso639-2/
- ISO 3166（国家代码）的维护机构为：

  ISO 3166 Maintenance Agency Secretariat（ ISO 3166 维护机构秘书处）
  c/o DIN Deutsches Institut fuer Normung（ c/o DIN 德国标准化研究所）
  Burggrafenstrasse 6
  Postfach 1107
  D-10787 Berlin
  Germany
  电话：+49 30 26 01 320
  传真：+49 30 26 01 231
  URL：http://www.din.de/gremien/nas/nabd/iso3166ma/ 601

附录H

MIME字符集注册表

本附录描述了由 IANA 维护的 MIME 字符集注册表。表 H-1 给出了注册表中字符集的格式化表格。

H.1　MIME字符集注册表

MIME 字符集标记是由 IANA（http://www.iana.org/numbers.htm）注册的。字符集注册表是各项记录的平面文件文本数据库。每条记录中都包含一个字符集名称、参考引用、唯一的 MIB 号、一段资源描述和别名列表。名字或别名可能会被标识为“首选 MIME 名”。

下面是 US-ASCII 的记录：

```
Name: ANSI_X3.4-1968 [RFC1345, KXS2]^
MIBenum: 3
Source: ECMA registry
Alias: iso-ir-6
Alias: ANSI_X3.4-1986
Alias: ISO_646.irv:1991
Alias: ASCII
Alias: ISO646-US
Alias: US-ASCII (preferred MIME name)
Alias: us
Alias: IBM367
Alias: cp367
Alias: csASCII
```

602 RFC 2978（http://www.ietf.org/rfc/rfc2978.txt）记录了通过 IANA 注册字符集的过程。

H.2　首选MIME名

在编写本书时已注册的 235 个字符集中，只有 20 个包含了“首选 MIME 名”，即电子邮件和 Web 应用程序使用的常见字符集。其中包括：

Big5	EUC-JP	EUC-KR
GB2312	ISO-2022-JP	ISO-2022-JP-2
ISO-2022-KR	ISO-8859-1	ISO-8859-2
ISO-8859-3	ISO-8859-4	ISO-8859-5
ISO-8859-6	ISO-8859-7	ISO-8859-8
ISO-8859-9	ISO-8859-10	KOI8-R
Shift-JIS	US-ASCII	

H.3 已注册字符集

表 H-1 列出了 2001 年 3 月字符集注册表的内容。更多与此表内容有关的信息请直接参阅 http://www.iana.org。

表H-1　IANA MIME字符集标记

字符集标记	别　　名	描　　述	参考文献
US-ASCII	ANSI_X3.4-1968、iso-ir-6、ANSI_X3.4-1986、ISO_646.irv:1991、ASCII、ISO646-US、us、IBM367、cp367、csASCII	ECMA 注册表	RFC1345、KXS2
ISO-10646-UTF-1	csISO10646UTF1	通用传输格式（1）——这是以 ASCII-7 为子集的多字节编码；不存在字节顺序问题	
ISO_646.basic:1983	ref、csISO646basic1983	ECMA 注册表	RFC1345、KXS2
INVARIANT	csINVARIANT		RFC1345、KXS2
ISO_646.irv:1983	iso-ir-2、irv、csISO2IntlRefVersion	ECMA 注册表	RFC1345、KXS2
BS_4730	iso-ir-4、ISO646-GB、gb、uk、csISO4UnitedKingdom	ECMA 注册表	RFC1345、KXS2
NATS-SEFI	iso-ir-8-1、csNATSSEFI	ECMA 注册表	RFC1345、KXS2
NATS-SEFI-ADD	iso-ir-8-2、csNATSSEFIADD	ECMA 注册表	RFC1345、KXS2
NATS-DANO	iso-ir-9-1、csNATSDANO	ECMA 注册表	RFC1345、KXS2
NATS-DANO-ADD	iso-ir-9-2、csNATSDANOADD	ECMA 注册表	RFC1345、KXS2　603
SEN_850200_B	iso-ir-10、FI、ISO646-FI、ISO646-SE、se、csISO10Swedish	ECMA 注册表	RFC1345、KXS2
SEN_850200_C	iso-ir-11、ISO646-SE2、se2、csISO11SwedishForNames	ECMA 注册表	RFC1345、KXS2
KS_C_5601-1987	iso-ir-149、KS_C_5601-1989、KSC_5601、korean、csKSC56011987	ECMA 注册表	RFC1345、KXS2
ISO-2022-KR	csISO2022KR	RFC 1557（还可参见 KS_C_5601-1987）	RFC1557、Choi
EUC-KR	csEUCKR	RFC 1557（还可参见 KS_C_5861-1992）	RFC1557、Choi

（续）

字符集标记	别　　名	描　　述	参考文献
ISO-2022-JP	csISO2022JP	RFC 1468（还可参见 RFC 2237）	RFC1468、Murai
ISO-2022-JP-2	csISO2022JP2	RFC 1554	RFC1554、Ohta
ISO-2022-CN		RFC 1922	RFC1922
ISO-2022-CN-EXT		RFC 1922	RFC1922
JIS_C6220-1969-jp	JIS_C6220-1969、iso-ir-13、katakana、x0201-7、csISO13JISC6220jp	ECMA 注册表	RFC1345、KXS2
JIS_C6220-1969-ro	iso-ir-14、jp、ISO646-JP、csISO14JISC6220ro	ECMA 注册表	RFC1345、KXS2
IT	iso-ir-15、ISO646-IT、csISO15Italian	ECMA 注册表	RFC1345、KXS2
PT	iso-ir-16、ISO646-PT、csISO16Portuguese	ECMA 注册表	RFC1345、KXS2
ES	iso-ir-17、ISO646-ES、csISO17Spanish	ECMA 注册表	RFC1345、KXS2
greek7-old	iso-ir-18、csISO18Greek7Old	ECMA 注册表	RFC1345、KXS2
latin-greek	iso-ir-19、csISO19LatinGreek	ECMA 注册表	RFC1345、KXS2
DIN_66003	iso-ir-21、de、ISO646-DE、csISO21German	ECMA 注册表	RFC1345、KXS2
NF_Z_62-010_(1973)	iso-ir-25、ISO646-FR1、csISO25French	ECMA 注册表	RFC1345、KXS2
Latin-greek-1	iso-ir-27、csISO27LatinGreek1	ECMA 注册表	RFC1345、KXS2
ISO_5427	iso-ir-37、csISO5427Cyrillic	ECMA 注册表	RFC1345、KXS2
JIS_C6226-1978	iso-ir-42、csISO42JISC62261978	ECMA 注册表	RFC1345、KXS2
BS_viewdata	iso-ir-47、csISO47BSViewdata	ECMA 注册表	RFC1345、KXS2
INIS	iso-ir-49、csISO49INIS	ECMA 注册表	RFC1345、KXS2
INIS-8	iso-ir-50、csISO50INIS8	ECMA 注册表	RFC1345、KXS2
INIS-cyrillic	iso-ir-51、csISO51INISCyrillic	ECMA 注册表	RFC1345、KXS2
ISO_5427:1981	iso-ir-54、ISO5427Cyrillic1981	ECMA 注册表	RFC1345、KXS2
ISO_5428:1980	iso-ir-55、csISO5428Greek	ECMA 注册表	RFC1345、KXS2
GB_1988-80	iso-ir-57、cn、ISO646-CN、csISO57GB1988	ECMA 注册表	RFC1345、K5、KXS2

604

（续）

字符集标记	别　　名	描　　述	参考文献
GB_2312-80	iso-ir-58、chinese、csISO58GB231280	ECMA 注册表	RFC1345、KXS2
NS_4551-1	iso-ir-60、ISO646-NO、no、csISO60DanishNorwegian、csISO60Norwegian1	ECMA 注册表	RFC1345、KXS2
NS_4551-2	ISO646-NO2、iso-ir-61、no2、csISO61Norwegian2	ECMA 注册表	RFC1345、KXS2
NF_Z_62-010	iso-ir-69、ISO646-FR、fr、csISO69French	ECMA 注册表	RFC1345、KXS2
videotex-suppl	iso-ir-70、csISO70VideotexSupp1	ECMA 注册表	RFC1345、KXS2
PT2	iso-ir-84、ISO646-PT2、csISO84Portuguese2	ECMA 注册表	RFC1345、KXS2
ES2	iso-ir-85、ISO646-ES2、csISO85Spanish2	ECMA 注册表	RFC1345、KXS2
MSZ_7795.3	iso-ir-86、ISO646-HU、hu、csISO86Hungarian	ECMA 注册表	RFC1345、KXS2
JIS_C6226-1983	iso-ir-87、x0208、JIS_X0208-1983、csISO87JISX0208	ECMA 注册表	RFC1345、KXS2
greek7	iso-ir-88、csISO88Greek7	ECMA 注册表	RFC1345、KXS2
ASMO_449	ISO_9036、arabic7、iso-ir-89、csISO89ASMO449	ECMA 注册表	RFC1345、KXS2
iso-ir-90	csISO90	ECMA 注册表	RFC1345、KXS2
JIS_C6229-1984-a	iso-ir-91、jp-ocr-a、csISO91JISC62291984a	ECMA 注册表	RFC1345、KXS2
JIS_C6229-1984-b	iso-ir-92、ISO646-JP-OCR-B、jp-ocr-b、csISO92JISC62991984b	ECMA 注册表	RFC1345、KXS2
JIS_C6229-1984-b-add	iso-ir-93、jp-ocr-b-add、csISO93JIS62291984badd	ECMA 注册表	RFC1345、KXS2
JIS_C6229-1984-hand	iso-ir-94、jp-ocr-hand、csISO94JIS62291984hand	ECMA 注册表	RFC1345、KXS2
JIS_C6229-1984-hand-add	iso-ir-95、jp-ocr-hand-add、csISO95JIS62291984handadd	ECMA 注册表	RFC1345、KXS2
JIS_C6229-1984-kana	iso-ir-96、csISO96JISC62291984kana	ECMA 注册表	RFC1345、KXS2
ISO_2033-1983	iso-ir-98、e13b、csISO2033	ECMA 注册表	RFC1345、KXS2

605

（续）

字符集标记	别　　名	描　　述	参考文献
ANSI_X3.110-1983	iso-ir-99、CSA_T500-1983、NAPLPS、csISO99NAPLPS	ECMA 注册表	RFC1345、KXS2
ISO-8859-1	ISO_8859-1:1987、iso-ir-100、ISO_8859-1、latin1、l1、IBM819、CP819、csISOLatin1	ECMA 注册表	RFC1345、KXS2
ISO-8859-2	ISO_8859-2:1987、iso-ir-101、ISO_8859-2、latin2、l2、csISOLatin2	ECMA 注册表	RFC1345、KXS2
T.61-7bit	iso-ir-102、csISO102T617bit	ECMA 注册表	RFC1345、KXS2
T.61-8bit	T.61、iso-ir-103、csISO103T618bit	ECMA 注册表	RFC1345、KXS2
ISO-8859-3	ISO_8859-3:1988、iso-ir-109、ISO_8859-3、latin3、l3、csISOLatin3	ECMA 注册表	RFC1345、KXS2
ISO-8859-4	ISO_8859-4:1988、iso-ir-110、ISO_8859-4、latin4、l4、csISOLatin4	ECMA 注册表	RFC1345、KXS2
ECMA-cyrillic	iso-ir-111、csISO111ECMACyrillic	ECMA 注册表	RFC1345、KXS2
CSA_Z243.4-1985-1	iso-ir-121、ISO646-CA、csa7-1、ca、csISO121Canadian1	ECMA 注册表	RFC1345、KXS2
CSA_Z243.4-1985-2	iso-ir-122、ISO646-CA2、csa7-2、csISO122Canadian2	ECMA 注册表	RFC1345、KXS2
CSA_Z243.4-1985-gr	iso-ir-123、csISO123CSAZ24341985gr	ECMA 注册表	RFC1345、KXS2
ISO-8859-6	ISO_8859-6:1987、iso-ir-127、ISO_8859-6、ECMA-114、ASMO-708、arabic、csISOLatinArabic	ECMA 注册表	RFC1345、KXS2
ISO_8859-6-E	csISO88596E	RFC 1556	RFC1556、IANA
ISO_8859-6-I	csISO88596I	RFC 1556	RFC1556、IANA
ISO-8859-7	ISO_8859-7:1987、iso-ir-126、ISO_8859-7、ELOT_928、ECMA-118、greek、greek8、csISOLatinGreek	ECMA 注册表	RFC1947、RFC1345、KXS2
T.101-G2	iso-ir-128、csISO128T101G2	ECMA 注册表	RFC1345、KXS2
ISO-8859-8	ISO_8859-8:1988、iso-ir-138、ISO_8859-8、hebrew、csISOLatinHebrew	ECMA 注册表	RFC1345、KXS2

（续）

字符集标记	别　　名	描　　述	参考文献
ISO_8859-8-E	csISO88598E	RFC 1556	RFC1556、Nussbacher 606
ISO_8859-8-I	csISO88598I	RFC 1556	RFC1556、Nussbacher
CSN_369103	iso-ir-139、csISO139CSN369103	ECMA 注册表	RFC1345、KXS2
JUS_I.B1.002	iso-ir-141、ISO646-YU、js、yu、csISO141JUSIB1002	ECMA 注册表	RFC1345、KXS2
ISO_6937-2-add	iso-ir-142、csISOTextComm	ECMA 注册表及 ISO 6937-2:1983	RFC1345、KXS2
IEC_P27-1	iso-ir-143、csISO143IECP271	ECMA 注册表	RFC1345、KXS2
ISO-8859-5	ISO_8859-5:1988、iso-ir-144、ISO_8859-5、cyrillic、csISOLatinCyrillic	ECMA 注册表	RFC1345、KXS2
JUS_I.B1.003-serb	iso-ir-146、serbian、csISO146Serbian	ECMA 注册表	RFC1345、KXS2
JUS_I.B1.003-mac	macedonian、iso-ir-147、csISO147Macedonian	ECMA 注册表	RFC1345、KXS2
ISO-8859-9	ISO_8859-9:1989、iso-ir-148、ISO_8859-9、latin5、l5、csISOLatin5	ECMA 注册表	RFC1345、KXS2
greek-ccitt	iso-ir-150、csISO150、csISO150GreekCCITT	ECMA 注册表	RFC1345、KXS2
NC_NC00-10:81	cuba、iso-ir-151、ISO646-CU、csISO151Cuba	ECMA 注册表	RFC1345、KXS2
ISO_6937-2-25	iso-ir-152、csISO6937Add	ECMA 注册表	RFC1345、KXS2
GOST_19768-74	ST_SEV_358-88、iso-ir-153、csISO153GOST1976874	ECMA 注册表	RFC1345、KXS2
ISO_8859-supp	iso-ir-154、latin1-2-5、csISO8859Supp	ECMA 注册表	RFC1345、KXS2
ISO_10367-box	iso-ir-155、csISO10367Box	ECMA 注册表	RFC1345、KXS2
ISO-8859-10	iso-ir-157、l6、ISO_8859-10:1992、csISOLatin6、latin6	ECMA 注册表	RFC1345、KXS2
latin-lap	lap、iso-ir-158、csISO158Lap	ECMA 注册表	RFC1345、KXS2
JIS_X0212-1990	x0212、iso-ir-159、csISO159JISX02121990	ECMA 注册表	RFC1345、KXS2

（续）

字符集标记	别　名	描　述	参考文献
DS_2089	DS2089、ISO646-DK、dk、csISO646Danish	丹麦标准，DS 2089，1974 年 2 月	RFC1345、KXS2
us-dk	csUSDK		RFC1345、KXS2
dk-us	csDKUS		RFC1345、KXS2
607 JIS_X0201	X0201、csHalfWidthKatakana	JIS X 0201-1976——只有 1 个字节，等价于（与 ASCII 类似的）JIS/Roman 加上 8 位半角片假名	RFC1345、KXS2
KSC5636	ISO646-KR、csKSC5636		RFC1345、KXS2
ISO-10646-UCS-2	csUnicode	2 个 8 位字节的基本多语种平台，又名 Unicode——这需要指定网络字节序。标准中并未对其进行详细说明（它是个 16 位的整数空间）	
ISO-10646-UCS-4	csUCS4	完整的代码空间（与字节序的注释相同，是 31 位的数字）	
DEC-MCS	dec、csDECMCS	VAX/VMS 用户手册，订单号：AI-Y517A-TE，1986 年 4 月	RFC1345、KXS2
hp-roman8	roman8、r8、csHPRoman8	LaserJet IIP 打印机用户手册，惠普部件编号 33471-90901，惠普，1989 年 6 月	HP-PCL5、RFC1345、KXS2
macintosh	mac、csMacintosh	Unicode 标准 v1.0，ISBN 号 0201567881，1991 年 10 月	RFC1345、KXS2
IBM037	cp037、ebcdic-cp-us、ebcdic-cp-ca、ebcdic-cp-wt、ebcdic-cp-nl、csIBM037	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM038	EBCDIC-INT、cp038、csIBM038	IBM 3174 字符集参考文献，GA27-3831-02，1990 年 3 月	RFC1345、KXS2

（续）

字符集标记	别　　名	描　　述	参考文献
IBM273	CP273、csIBM273	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM274	EBCDIC-BE、CP274、csIBM274	IBM 3174 字符集参考文献，GA27-3831-02，1990 年 3 月	RFC1345、KXS2
IBM275	EBCDIC-BR、cp275、csIBM275	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM277	EBCDIC-CP-DK、EBCDIC-CP-NO、csIBM277	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM278	CP278、ebcdic-cp-fi、ebcdic-cp-se、csIBM278	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM280	CP280、ebcdic-cp-it、csIBM280	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM281	EBCDIC-JP-E、cp281、csIBM281	IBM 3174 字符集参考文献，GA27-3831-02，1990 年 3 月	RFC1345、KXS2
IBM284	CP284、ebcdic-cp-es、csIBM284	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM285	CP285、ebcdic-cp-gb、csIBM285	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM290	cp290、EBCDIC-JP-kana、csIBM290	IBM 3174 字符集参考文献、GA27-3831-02，1990 年 3 月	RFC1345、KXS2
IBM297	cp297、ebcdic-cp-fr、csIBM297	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2 608
IBM420	cp420、ebcdic-cp-ar1、csIBM420	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月，IBM NLS RM p 11-11	RFC1345、KXS2

（续）

字符集标记	别　　名	描　　述	参考文献
IBM423	cp423、ebcdic-cp-gr、csIBM423	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM424	cp424、ebcdic-cp-he、csIBM424	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM437	cp437、437、csPC8CodePage437	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM500	CP500、ebcdic-cp-be、ebcdic-cp-ch、csIBM500	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM775	cp775、csPC775Baltic	HP PCL 5 Comparison Guide（P/N 5021-0329）B-13 页，1996 年	HP-PCL5
IBM850	cp850、850、csPC850Multilingual	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM851	cp851、851、csIBM851	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM852	cp852、852、csPCp852	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM855	cp855、855、csIBM855	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM857	cp857、857、csIBM857	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM860	cp860、860、csIBM860	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM861	cp861、861、cp-is、csIBM861	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2

（续）

字符集标记	别　　名	描　　述	参考文献
IBM862	cp862、862、csPC862LatinHebrew	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM863	cp863、863、csIBM863	IBM 键盘布局和代码页，PN 07G4586，1991 年 6 月	RFC1345、KXS2
IBM864	cp864、csIBM864	IBM 键盘布局和代码页，PN 07G4586，1991 年 6 月	RFC1345、KXS2
IBM865	cp865、865、csIBM865	IBM DOS 3.3 参考文献（删节版），94X9575，1987 年 2 月	RFC1345、KXS2
IBM866	cp866、866、csIBM866	IBM NLDG 第二卷 SE09-8002-03，1994 年 8 月	Pond
IBM868	CP868、cp-ar、csIBM868	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM869	cp869、869、cp-gr、csIBM869	IBM 键盘布局和代码页，PN 07G4586，1991 年 6 月	RFC1345、KXS2
IBM870	CP870、ebcdic-cp-roece、ebcdic-cp-yu、csIBM870	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM871	CP871、ebcdic-cp-is、csIBM871	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM880	cp880、EBCDIC-Cyrillic、csIBM880	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM891	cp891、csIBM891	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM903	cp903、csIBM903	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2

609

（续）

字符集标记	别　　名	描　　述	参考文献
IBM904	cp904、904、csIBBM904	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM905	CP905、ebcdic-cp-tr、csIBM905	IBM 3174 字符集参考文献、GA27-3831-02，1990 年 3 月	RFC1345、KXS2
IBM918	CP918、ebcdic-cp-ar2、csIBM918	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
IBM1026	CP1026、csIBM1026	IBM NLS RM 第二卷 SE09-8002-01，1990 年 3 月	RFC1345、KXS2
EBCDIC-AT-DE	csIBMEBCDICATDE	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-AT-DE-A	csEBCDICATDEA	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-CA-FR	csEBCDICCAFR	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-DK-NO	csEBCDICDKNO	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-DK-NO-A	csEBCDICDKNOA	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-FI-SE	csEBCDICFISE	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-FI-SE-A	csEBCDICFISEA	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-FR	csEBCDICFR	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2

（续）

字符集标记	别　　名	描　　述	参考文献
EBCDIC-IT	csEBCDICIT	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-PT		IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-ES	csEBCDICES	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-ES-A	csEBCDICESA	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2 610
EBCDIC-ES-S	csEBCDICESS	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-UK	csEBCDICUK	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
EBCDIC-US	csEBCDICUS	IBM 3270 字符集参考文献第 10 章，GA27-2837-9，1987 年 4 月	RFC1345、KXS2
UNKNOWN-8BIT	csUnknown8BiT		RFC1428
MNEMONIC	csMnemonic	RFC 1345，也称作“mnemonic+ascii+38”	RFC1345、KXS2
MNEM	csMnem	RFC 1345，也称作“mnemonic+ascii+ 8200”	RFC1345、KXS2
VISCII	csVISCII	RFC 1456	RFC1456
VIQR	csVIQR	RFC 1456	RFC1456
KOI8-R	csKOI8R	RFC 1489，基于 GOST-19768-74、ISO-6937/8、INIS-Cyrillic、ISO-5427	RFC1489
KOI8-U		RFC 2319	RFC2319
IBM00858	CCSID00858、CP00858、PC-Multilingual-850+euro	IBM（参见 .../assignments/character-set-info/IBM00858）[Mahdi]	

（续）

字符集标记	别　　名	描　　述	参考文献
IBM00924	CCSID00924、CP00924、ebcdic-Latin9—euro	IBM（参见 .../assignments/character-set-info/IBM00924）[Mahdi]	
IBM01140	CCSID01140、CP01140、ebcdic-us-37+euro	IBM（参见 .../assignments/character-set-info/IBM01140）[Mahdi]	
IBM01141	CCSID01141、CP01141、ebcdic-de-273+euro	IBM（参见 .../assignments/character-set-info/IBM01141）[Mahdi]	
IBM01142	CCSID01142、CP01142、ebcdic-dk-277+euro、ebcdic-no-277+euro	IBM（参见 .../assignments/character-set-info/IBM01142）[Mahdi]	
IBM01143	CCSID01143、CP01143、ebcdic-fi-278+euro、ebcdic-se-278+euro	IBM（参见 .../assignments/character-set-info/IBM01143）[Mahdi]	
IBM01144	CCSID01144、CP01144、ebcdic-it-280+euro	IBM（参见 .../assignments/character-set-info/IBM01144）[Mahdi]	
IBM01145	CCSID01145、CP01145、ebcdic-es-284+euro	IBM（参见 .../assignments/character-set-info/IBM01145）[Mahdi]	
IBM01146	CCSID01146、CP01146、ebcdic-gb-285+euro	IBM（参见 .../assignments/character-set-info/IBM01146）[Mahdi]	
IBM01147	CCSID01147、CP01147、ebcdic-fr-297+euro	IBM（参见 .../assignments/character-set-info/IBM01147）[Mahdi]	
611 IBM01148	CCSID01148、CP01148、ebcdic-international-500+euro	IBM（参见 .../assignments/character-set-info/IBM01148）[Mahdi]	
IBM01149	CCSID01149、CP01149、ebcdic-is-871+euro	IBM（参见 .../assignments/character-set-info/IBM01149）[Mahdi]	
Big5-HKSCS	None	（参见 .../assignments/character-set-info/Big5-HKSCS）[Yick]	

（续）

字符集标记	别　　名	描　　述	参考文献
UNICODE-1-1	csUnicode11	RFC 1641	RFC1641
SCSU	None	SCSU（参见 .../assignments/character-set-info/SCSU）[Scherer]	
UTF-7	None	RFC 2152	RFC2152
UTF-16BE	None	RFC 2781	RFC2781
UTF-16LE	None	RFC 2781	RFC2781
UTF-16	None	RFC 2781	RFC2781
UNICODE-1-1-UTF-7	csUnicode11UTF7	RFC 1642	RFC1642
UTF-8		RFC 2279	RFC2279
iso-8859-13		ISO（参见 ...assignments/character-set-info/iso-8859-13）[Tumasonis]	
iso-8859-14	iso-ir-199、ISO_8859-14:1998、ISO_8859-14、latin8、iso-celtic、l8	ISO（参见 ...assignments/character-set-info/iso-8859-14）[Simonsen]	
ISO-8859-15	ISO_8859-15	ISO	
JIS_Encoding	csJISEncoding	JIS X 0202-1991；使用 ISO 2022 转义序列来切换代码集，如 JIS X 0202-1991 所述	
Shift_JIS	MS_Kanji、csShiftJIS	这个字符集是 csHalf WidthKatakana 的扩展——它在 JIS X 0208 中添加了图像字符。CCS 是 JIS X0201:1997 和 JIS X0208:1997。JISX0208: 1997 的附录 1 给出了完整的定义。可以将此字符集用于顶级媒体类型“text”	

（续）

字符集标记	别　　名	描　　述	参考文献
EUC-JP	Extended_UNIX_Code_Packed_Format_for_Japanese、csEUCPkdFmtJapanese	由 OSF、UNIX 国际和 UNIX 系统太平洋实验室制定的标准。使用 ISO 2022 规则来挑选代码集。代码集 0：US-ASCII（单 7 位的字节集）；代码集 1：JIS X0208-1990（双 8 位的字节集），这两个字节都限制在 A0 ~ FF 之间；代码集 2：半角片假名（单 7 位的字节集）需要将 SS2 作为字符前缀；代码集 3：JIS X0212-1990（双 7 位的字节集），两个字节都要限制在 A0 ~ FF 之间，需要将 SS3 作为字符前缀	
Extended_UNIX_Code_Fixed_Width_for_Japanese	csEUCFixWidJapanese	在日本使用。每个字符都是 2 个字节。代码集 0：US-ASCII（单 7 位的字节集），第一个字节 00，第二个字节 20 ~ 7E；代码集 1：JIS X0208-1990（双 7 位的字节集），两个字节都限制在 A0 ~ FF 之间；代码集 2：半角片假名（单 7 位的字节集），第一个字节 00，第二个字节 A0 ~ FF；代码集 3：JIS X0212-1990（双 7 位字节集），第一个字节限制在 A0 ~ FF 之间，第二个字节限制在 21 ~ 7E 之间	
ISO-10646-UCS-Basic	csUnicodeASCII	Unicode 的 ASCII 子集。基本拉丁字母为集 1。参见 ISO 10646，附录 A	
ISO-10646-Unicode-Latin1	csUnicodeLatin1、ISO-10646	Unicode 的 ISO Latin-1 子集。基本拉丁字母和 Latin-1。增补为集 1 及集 2。参见 ISO 10646，附录 A 以及 RFC 1815	

612

（续）

字符集标记	别　　名	描　　述	参考文献
ISO-10646-J-1		ISO 10646 日语。参见 RFC 1815	
ISO-Unicode-IBM-1261	csUnicodeIBM1261	IBM Latin-2、-3、-5、扩展表示集，GCSGID: 1261	
ISO-Unicode-IBM-1268	csUnidoceIBM1268	IBM Latin-4 扩展表示集，GCSGID: 1268	
ISO-Unicode-IBM-1276	csUnicodeIBM1276	IBM 西里尔希腊语扩展表示集，GCSGID: 1276	
ISO-Unicode-IBM-1264	csUnicodeIBM1264	IBM 阿拉伯语扩展表示集，GCSGID: 1264	
ISO-Unicode-IBM-1265	csUnicodeIBM1265	IBM 希伯来语扩展表示集，GCSGID: 1265	
ISO-8859-1-Windows-3.0-Latin-1	csWindows30Latin1	用于 Windows 3.0 的扩展 ISO 8859-1 Latin-1。PCL 符号集 ID：9U	HP-PCL5
ISO-8859-1-Windows-3.1-Latin-1	csWindows31Latin1	用于 Windows 3.1 的扩展 ISO 8859-1 Latin-1。PCL 符号集 ID：19U	HP-PCL5
ISO-8859-2-Windows-Latin-2	csWindows31Latin2	用于 Windows 3.1 的扩展 ISO 8859-2 Latin-2。PCL 符号集 ID：9E	HP-PCL5
ISO-8859-9-Windows-Latin-5	csWindows31Latin5	用于 Windows 3.1 的扩展 ISO 8859-9 Latin-5。PCL 符号集 ID：5T	HP-PCL5
Adobe-Standard-Encoding	csAdobeStandardEncoding	PostScript 语言参考手册。PCL 符号集 ID：10J	Adobe
Ventura-US	csVenturaUS	Ventura US-ASCII plus 字符通常用于出版业，比如在 A0 ~ FF 范围之间的段落标记、版权、注册、商标、分节、剑号及双剑号。PCL 符号集 ID：14J	HP-PCL5

613

（续）

字符集标记	别　　名	描　　述	参考文献
Ventura-International	csVenturaInternational	Ventura International ASCII plus 编码字符与 Roman8 类似。PCL 符号集 ID：13J	HP-PCL5
PC8-Danish-Norwegian	csPC8DanishNorwegian	丹麦挪威使用的 PC 丹麦挪威 8 位 PC 集。PCL 符号集 ID：11U	HP-PCL5
PC8-Turkish	csPC8Turkish	PC 拉丁土耳其。PCL 符号集：9T	HP-PCL5
IBM-Symbols	csIBMSymbols	表示集，CPGID: 259	IBM-CIDT
IBM-Thai	csIBMThai	表示集，CPGID: 838	IBM-CIDT
HP-Legal	csHPLegal	PCL 5 Comparison Guide，惠普，惠普部件编号 5961-0510，1992 年 10 月。PCL 符号集 ID：1U	HP-PCL5
HP-Pi-font	csHPPiFont	PCL 5 Comparison Guide，惠普，惠普部件编号 5961-0510，1992 年 10 月。PCL 符号集 ID：15U	HP-PCL5
HP-Math8	csHPMath8	PCL 5 Comparison Guide，惠普，惠普部件编号 5961-0510，1992 年 10 月。PCL 符号集 ID：8M	HP-PCL5
Adobe-Symbol-Encoding	csHPPSMath	PostScript 语言参考手册。PCL 符号集 ID：5M	Adobe
HP-DeskTop	csHPDesktop	PCL 5 Comparison Guide，惠普，惠普部件编号 5961-0510，1992 年 10 月。PCL 符号集 ID：7J	HP-PCL5
Ventura-Math	csVenturaMath	PCL 5 Comparison Guide，惠普，惠普部件编号 5961-0510，1992 年 10 月。PCL 符号集 ID：6M	HP-PCL5
Microsoft-Publishing	csMicrosoftPublishing	PCL 5 Comparison Guide，惠普，惠普部件编号 5961-0510，1992 年 10 月。PCL 符号集 ID：6J	HP-PCL5

（续）

字符集标记	别　　名	描　　述	参考文献
Windows-31J	csWindows31J	日语版 Windows。是对 Shift_JIS 的进一步扩展，以包含 NEC 特殊字符（第 13 行）、NEC 选择的 IBM 扩展（第 89 ～ 92 行）、以及 IBM 扩展（第 115 ～ 119 行）。CCS 包含 JIS X0201:1997、JIS X0208:1997 和这些扩展。可将此字符集用于顶级媒体类型“text”，但用途有限，或仅用于特殊目的（参见 RFC 2278）。PCL 字符集 ID：19K	
GB2312	csGB2312	混合了 1 字节、2 字节集的中华人民共和国（PRC）汉语：20-7E 为 1 字节 ASCII；A1-FE 为 2 字节 PRC 汉字。参见 GB 2312-80。PCL 符号集 ID：18C	
Big5	csBig5	中国台湾省使用的汉语多字节集。PCL 符号集 id：18T	
windows-1250		微软（参见 .../character-set-info/windows-1250）[Lazhintseva]	
windows-1251		微软（参见 .../character-set-info/windows-1251）[Lazhintseva]	
windows-1252		微软（参见 .../character-set-info/windows-1252）[Wendt]	
windows-1253		微软（参见 .../character-set-info/windows-1253）[Lazhintseva]	
windows-1254		微软（参见 .../character-set-info/windows-1254）[Lazhintseva]	

614

（续）

字符集标记	别　　名	描　　述	参考文献
windows-1255		微软（参见 .../character-set-info/windows-1255）[Lazhintseva]	
windows-1256		微软（参见 .../character-set-info/windows-1256）[Lazhintseva]	
windows-1257		微软（参见 .../character-set-info/windows-1257）[Lazhintseva]	
windows-1258		微软（参见 .../character-set-info/windows-1258）[Lazhintseva]	
TIS-620		TISI，Thai Industrial Standards Institute（泰国工业标准协会）	[Tantsetthi]
HZ-GB-2312		RFC 1842、RFC 1843 [RFC1842、RFC1843]	

615

索引

索引中的页码为英文原书的页码，与本书页边标注的页码一致。

符号

数字

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

关于作者

David Gourley 是 Endeca 的首席技术官，负责 Endeca 产品的研究及开发。Endeca 开发的因特网及内部网络信息访问解决方案为企业级数据的导航及研究提供了一些新的方式。在到 Endeca 工作之前，David 是 Inktomi 基础工程组的一员，帮助开发了 Inktomi 的因特网搜索数据库，也是 Inktomi 的 Web 缓存产品的主要开发者。

David 在加州大学伯克利分校获得了计算机科学的学士学位，还拥有 Web 技术方面的几项专利。

Brian Totty 最近出任了 Inktomi 公司（这是 1996 年他参与建立的一家公司）研发部副总裁，在公司中他负责 Web 缓存、流媒体及因特网搜索技术的研发工作。他曾是 Silicon Graphics 公司的一名研究员，为高性能网络和超级计算机系统设计软件并对其进行优化。在那之前，他是苹果计算机公司高级技术组的一名工程师。

Brian 在伊利诺伊大学 Urbana-Champaign 分校获得了计算机科学的博士学位，在 MIT 获得了计算机科学及电子工程的学士学位，以及计算机系统研究的 Organick 奖。他还在加州大学分校开展并讲授了一些屡获殊荣的因特网技术方面的课程。

Marjorie Sayer 在 Inktomi 公司负责编写 Web 缓存方面的软件。在加州大学伯克利分校获得了数学硕士和博士学位之后，一直致力于数学课程的改革。从 1990 年开始致力于能量资源管理、并行系统软件、电话和网络方面的写作。

Sailu Reddy 目前在 Inktomi 公司负责嵌入式的性能增强型 HTTP 代理的开发。Sailu 从事复杂软件系统的开发已经有 12 年了，从 1995 年开始深入 Web 架构的研发工作。他是 Netscape 第一台 Web 服务器、Web 代理产品，以及后面几代产品的核心工程师。他具备 HTTP 应用程序、数据压缩技术、数据库引擎以及合作管理等

方面的技术经验。Sailu 在亚里桑那大学获得了信息系统的硕士学位并握有 Web 技术方面的多项专利。

Anshu Aggarwal 是 Inktomi 公司的工程总监。他管理 Inktomi 公司 Web 缓存产品的协议处理工程组，从 1997 年就开始参与 Inktomi 的 Web 技术设计工作。Anshu 在科罗拉多大学 Boulder 分校获得了计算机科学的硕士和博士学位，从事分布式多处理器的内存一致性技术研究。他还拥有电子工程的硕士和学士学位。Anshu 撰写了多篇技术论文，还拥有两项专利。

尾页

本书的封面设计得益于多方的参与，来自于读者意见、我们自己的揣摩以及各种发行渠道的反馈，共同的努力使它变得与众不同，既体现出我们诠释技术话题的独特手法，也使得有些原本枯燥的话题变得有趣。

封面上的动物叫多纹黄鼠（thirteen-lined ground squirrel，Spermophilus tridecemlineatus），在北美洲中部很常见。鼠如其名，沿着多纹黄鼠的背部有十三条条纹，上面有一行行的浅色斑点。它的颜色模式能让其融入周围的环境，以防捕食者发现它。多纹黄鼠是松鼠家族中的一员，松鼠家族中包括花栗鼠、地松鼠、树松鼠、土拨鼠和旱獭。它们的体型与东方花栗鼠类似，但比常见的灰松鼠小，平均长度大概有 11 英寸（加上一个 5 ～ 6 英寸的尾巴）。

多纹黄鼠会在十月份冬眠，并在三月末四月初苏醒过来。通常每只雌鼠会在每年五月产下一窝 7 ～ 10 只小鼠。小鼠会在四到五周大的时候离开地洞，并在六周的时候完全长成。它们喜欢在长有短草且排水良好的砂质开阔地或肥沃的土壤上打洞，它们不喜欢树木繁茂的地区——修剪过的草坪、高尔夫球场和公园是它们常见的栖息地。

地松鼠会引发一些问题，比如挖洞、挖出种下的种子，破坏菜园子。但它们同样也是几种捕食者的重要猎物，包括獾、草原狼、鹰、黄鼠狼和各种蛇，而且它们以很多有害的杂草、杂草种子及昆虫为食，会让人类直接受益。

本书的项目及文字编辑是 Rachel Wheeler。质量控制人员包括：Leanne Soylemez、Sarah Sherman 和 Mary Anne Weeks Mayo。其他辅助工作由 Derek Di Matteo 和 Brian Sawyer 完成。John Bickelhaupt 编写了索引部分。

本书封面是在 Edie Freedman 设计的一系列封面基础上完成的，并由 Ellie Volckhausen 完成具体的设计。封面的原始图片由 Lorrie LeJeune 绘制。封面布局是 Emma Colby 用 QuarkXPress 4.1 以 Adobe 的 ITC Garamond 字体绘制的。

David Futato 和 Melanie Wang 根据 David Futato 的一系列设计完成了内部版面的设计。Joe Wizda 用 FrameMaker 5.5.6 为制版准备了一些文件。书中的图片都是 Robert Romano 和 Jessamyn Read 用 Macromedia FreeHand 9 和 Adobe Photoshop 6 制作的。尾页是 Rachel Wheeler 编写的。

本书的在线版本是 Safari 产品组（John Chodacki、Becki Maisch 和 Madeleine Newell）用 Erik Ray、Benn Salter、John Chodacki 和 Jeff Liggett 编写并维护的一组 Frame-to-XML 转换及清理工具创建的。

Web Programming

O'REILLY®

HTTP权威指南



HTTP是Web客户端与服务器交互文档和信息时所使用的协议，是每个成功Web事务的幕后推手。众所周知，我们每天访问公司内部网络、搜索绝版书籍、研究统计信息时所使用的浏览器的核心就是HTTP。但HTTP的应用远不仅仅是浏览Web内容。由于HTTP既简单又普及，很多其他网络应用程序也选择了它，尤其是采用SOAP和XML-RPC这样的Web服务。

本书详细解释了HTTP协议，包括它是如何工作的，如何用它来开发基于Web的应用程序。但本书并不只介绍了HTTP，还探讨了HTTP有效工作所依赖的所有其他核心因特网技术。尽管HTTP是本书的中心内容，但本书的本质是理解Web的工作原理，以及如何将这些知识应用到Web编程和管理之中，主要涵盖HTTP的技术运作方式、产生动机、性能和目标以及一些相关技术问题。

本书是HTTP协议及相关Web技术方面的权威著作，主要内容包括：

- HTTP方法、首部以及状态码
- 优化代理和缓存的方法
- 设计Web机器人和爬虫的策略
- Cookies、认证以及安全HTTP
- 国际化及内容协商
- 重定向及负载平衡策略

本书由具有多年实践经验的专家编写，通过简洁、精确的语言和大量翔实的细节图解帮助读者形象地理解Web幕后所发生的事情，详细说明了Web上每条请求的实际运行情况。

要想高效地进行Web开发，所有Web程序员、管理员和应用程序开发者都应该熟悉HTTP。很多书籍只介绍了Web的使用方式，而本书则深入说明了Web的工作原理。

封面设计：Karen Montgomery　张健

图灵社区：www.ituring.com.cn
新浪微博：@图灵教育 @图灵社区
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

O'REILLY®
oreilly.com.cn

分类建议　计算机/网络技术/HTTP

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

ISBN 978-7-115-28148-7

ISBN 978-7-115-28148-7
定价：109.00元

欢迎加入

图灵社区

电子书发售平台

电子出版的时代已经来临，在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版的梦想。你可以联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者，这极大地降低了出版的门槛。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果有意翻译哪本图书，欢迎来社区申请。只要通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

读者交流平台

在图灵社区，读者可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。欢迎大家积极参与社区开展的访谈、审读、评选等多种活动，赢取银子，可以换书哦！